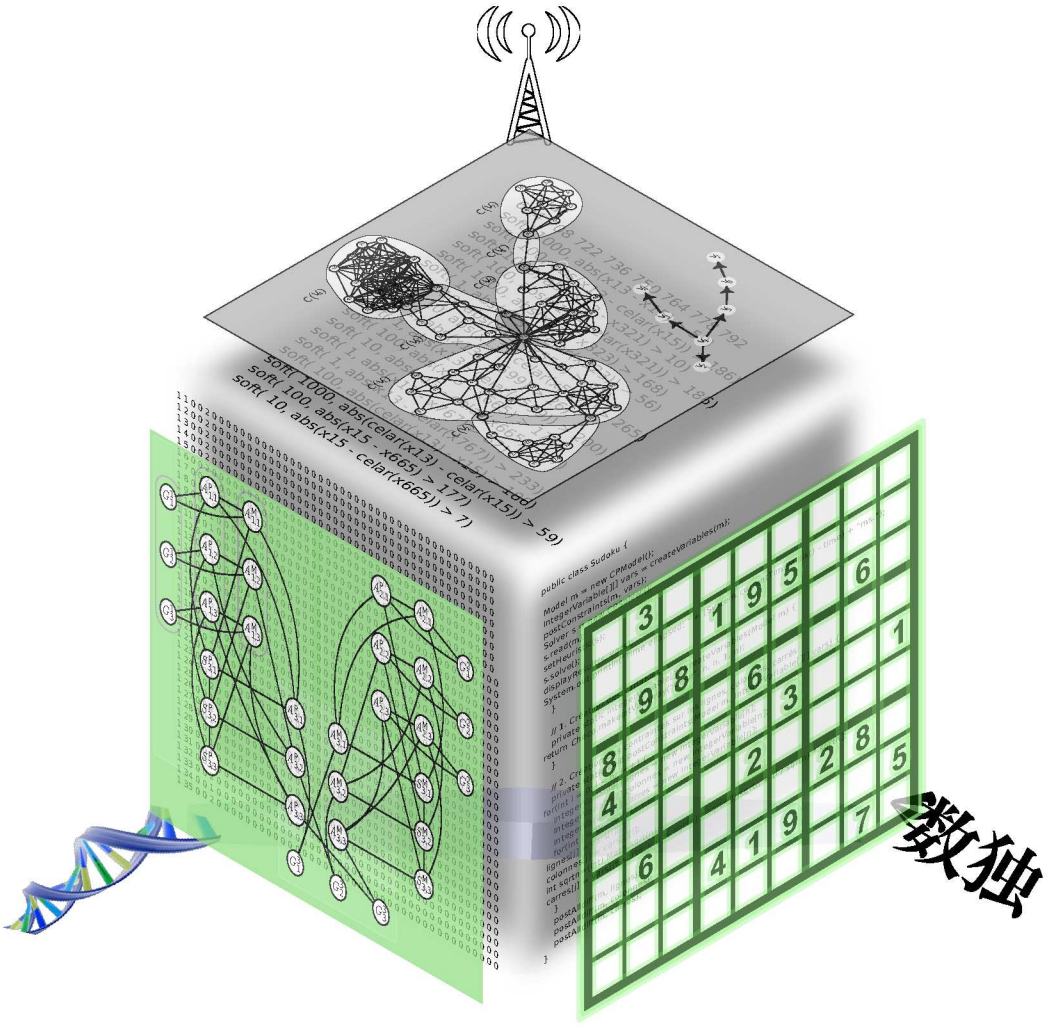
afpc jfpc 2012



Huitièmes Journées Francophones de Programmation par Contraintes

Toulouse, 22-24 mai 2012

Actes des Huitièmes Journées Francophones de Programmation par Contraintes

JFPC 2012

22-24 Mai 2012, Campus SUPAERO, ISAE, Toulouse, France

Présidents des journées

Christophe Garion, MARS-ISAE Toulouse
Marie-José Huguet, LAAS-CNRS, INSA Toulouse
Thomas Schiex, INRA Toulouse

Président du comité de programme

Simon de Givry, INRA Toulouse

Préface

La Programmation par Contraintes (ou PPC) permet de modéliser et de résoudre aisément des problèmes combinatoires de façon déclarative : « L'utilisateur décrit son problème, l'ordinateur le résout. » [Gene Freuder, *In pursuit of the Holy Grail*, Constraints 2(1), 1997]. Décrire ou modéliser son problème consiste à trouver les variables et les contraintes liants ces variables. On recherche alors une solution satisfaisant toutes les contraintes, ou dans le cas d'un problème d'optimisation, la meilleure solution satisfaisant les contraintes. Pour cela, le problème est transmis à un solveur qui va effectuer cette recherche.

Les origines de la PPC sont nombreuses et reflètent sa diversité actuelle : la satisfaction de contraintes en Intelligence Artificielle, les aspects langage issus de la Programmation en Logique par Contraintes, la Recherche Opérationnelle forment le cœur des influences.

Les Journées Francophones de Programmation par Contraintes (JFPC) sont avant tout placées sous le signe de la convivialité. Elles permettent à la communauté PPC francophone de se rencontrer et d'échanger. Les actes de cette conférence contiennent les dernières avancées de ce domaine présentées en langue française. Chercheurs, enseignants-chercheurs, étudiants et industriels se retrouvent pour communiquer et visualiser un vaste panorama de l'actualité du domaine, témoignant de la représentativité de la communauté francophone à l'échelon international. Les articles présentés sont soit originaux, soit ont été publiés l'an passé dans les grandes conférences internationales du domaine (IJCAI, AAAI, ECAI, CP, SAT, CPAIOR,...).

Les JFPC sont issues de la fusion des conférences JFPLC (Journées Francophones de Programmation en Logique et par Contraintes) nées en 1992 et des JNPC (Journées Nationales sur la résolution pratique des problèmes NP-complets) nées en 1994. Huitième édition de la série, les JFPC 2012 à Toulouse succèdent à celles de Lyon (2011), Caen (2010), Orléans (2009), Nantes (2008), Rocquencourt (2007), Nîmes (2006) et Lens (2005).

Les JFPC ont eu le plaisir de recevoir quarante-neuf soumissions, provenant de la communauté francophone en France, Algérie, Belgique, Cameroon, Chili, Egypte, Espagne, Hong Kong, Maroc et USA. Après une relecture soigneuse effectuée par trois ou quatre experts, le comité de programme en a sélectionné quarante-et-un pour une présentation orale. Les thèmes abordés concernent le filtrage et la propagation, les symétries, la complexité des algorithmes, l'optimisation, la recherche locale, la satisfaction booléenne, le parallélisme, la satisfaction de contraintes quantifiées, les CSP numériques, la planification, la fouille de données, les ASP, sans oublier les applications industrielles.

Les JFPC 2012 ont été organisées conjointement avec les JIAF, Journées d'Intelligence Artificielle Fondamentale et ont comme but de faire se rencontrer les deux communautés. Cette rencontre prend la forme d'une conférence invitée *Programmation Logique Inductive* (Christel Vrain, Céline Rouveirol) et d'un tutoriel *Approches déclaratives pour l'énumération de motifs intéressants* (Lhouari Nourine, Jean-Marc Petit, Lakhdar Saïs) communs. De plus, deux conférences invitées plus typées JFPC sont données par Thierry Petit sur le thème de l'optimisation en programmation par contraintes et par Christian Bessière sur la décomposition de contraintes globales.

Je remercie les auteurs des articles soumis pour leur participation à cet évènement annuel témoignant de la vitalité de notre domaine, les membres du comité de programme et les relecteurs additionnels pour leur travail sérieux et constructif, Marie-José Huguet, Christophe Garion, Thomas Schiex et l'ensemble des organisateurs qui ont tout mis en œuvre pour que la conférence soit un succès et enfin les sponsors sans qui l'évènement n'aurait pu avoir lieu : ACP, AFPC, CNRS, Cosytec, Ecole des Mines d'Albi, ENAC, GDR I3, INRA, INRIA, INSA Toulouse, IRIT, ISAE, LAAS-CNRS, Mairie de Toulouse, ONERA, Prométil et la Région Midi-Pyrénées.

Simon de Givry, président du comité de programme des JFPC 2012

Comité de programme

Nicolas Barnier	ENAC, Toulouse
Patrice Boizumault	GREYC, Caen
Lucas Bordeaux	Microsoft Cambridge, UK
Gilles Chabert	EMN/LINA, Nantes
Remi Coletta	LIRMM, Montpellier
Nadia Creignou	Aix-Marseille Université
Thi-Bich-Hanh Dao	LIFO, Orléans
Gilles Dequen	MIS, Amiens
Eric Fanchon	TIMC-IMAG, Grenoble
Helene Fargier	IRIT, Toulouse
Carmen Gervet	GUC, Le Caire, Égypte
Simon de Givry	INRA, Toulouse
Alexandre Goldsztejn	CNRS/LINA, Nantes
Arnaud Gotlieb	IRISA, Rennes
Laurent Granvilliers	LINA, Nantes
Emmanuel Hebrard	LAAS, Toulouse
Fred Hemery	CRIL, Lens
George Katsirelos	INRA, Toulouse
Frederic Lardeux	LERIA, Angers
Xavier Lorca	LINA, Nantes
Florent Madelaine	LIX, Paris
Laurent Michel	CSED - University of Connecticut, USA
Jean-Noël Monette	Uppsala University, Suède
Eric Monfroy	LINA, Nantes
Samba-Ndojh Ndiaye	LIRIS, Lyon
Richard Ostrowski	LSIS, Marseille
Cédric Pralet	ONERA, Toulouse
Philippe Refalo	ILOG, Sophia
Céline Robardet	INSA, Lyon
Louis-Martin Rousseau	Polytechnique, Montréal, Canada
Olivier Roussel	CRIL, Lens
Michel Rueher	I3S/CNRS - Université de Nice Sophia Antipolis
Lakhdar Saïs	CRIL, Lens
Pierre Schaus	n-Side, Louvain-la-Neuve, Belgique
Pierre Siegel	SNC, Toulon
Laurent Simon	LRI - Université Paris Sud 11, Orsay
Sylvain Soliman	INRIA, Rocquencourt
Igor Stéphan	LERIA, Angers
Elise Vareilles	EMAC, Albi
Philippe Vismara	LIRMM, Montpellier

Relecteurs additionnels

Allignol, Cyril
Barbanchon, Régis
Devendeville, Laure
Durand, Nicolas
Fages, François
Grandcolas, Stéphane
Hervieu, Aymeric
Jabbour, Said
Lazaar, Nadjib
Lefèvre, Claire
Legendre, Florian
Lhomme, Olivier
Métivier, Jean-Philippe
Narodytska, Nina
Olive, Frédéric
Prud'Homme, Charles
Siala, Mohamed
Trilling, Laurent
Vander-Swalmen, Pascal

Comité d'organisation

Cyril Allignol	CENA, Toulouse
Nicolas Barnier	ENAC, Toulouse
Caroline Becker	IRIT, Toulouse
Pierre Bisquert	IRIT, Toulouse
Sonia Cafieri	ENAC, Toulouse
Sylvie Doutre	IRIT, Toulouse
Florence Dupin de Saint-Cyr	IRIT, Toulouse
Hélène Fargier	IRIT, Toulouse
Paul Gaborit	EMAC, Albi
Christophe Garion	ISAE, Toulouse
Alexandre Gondran	ENAC, Toulouse
Emmanuel Hébrard	LAAS, Toulouse
Marie-José Huguet	LAAS, Toulouse
Dominique Longin	IRIT, Toulouse
Pierre Lopez	LAAS, Toulouse
Catherine Mancel	ENAC, Toulouse
Jérôme Mengin	IRIT, Toulouse
Laurent Pérussel	IRIT, Toulouse
Cédric Pralet	ONERA, Toulouse
Thomas Schiex	INRA, Toulouse
Elise Vareilles	EMAC, Albi
Gérard Verfaillie	ONERA, Toulouse

Table des matières

Contraintes globales et décompositions

Christian Bessiere
U. Montpellier, France
bessiere@lirmm.fr

La propagation de contraintes est un composant essentiel de la programmation par contraintes [4]. Les contraintes peuvent être d'arité fixe, c'est à dire définies pour un nombre donné de variables. Par exemple $|x-y| = z$ ou $x \neq y$. Mais les contraintes peuvent aussi être "globales", c'est à dire exprimer une propriété pouvant porter sur un nombre quelconque de variables. Par exemple `alldifferent` spécifie que toutes les variables impliquées, quel que soit leur nombre, doivent prendre des valeurs différentes [10]. Les contraintes d'arité fixe peuvent être propagées en temps polynomial avec un algorithme générique. Les contraintes globales ne peuvent pas être propagées en temps polynomial si on ne dispose pas d'un algorithme ad hoc. Malgré tout, depuis 15 ans, les contraintes globales se sont rendues indispensables parce que quand on dispose d'un algorithme pour les propager, elles permettent souvent une propagation forte [13].

Il existe plus de 300 contraintes globales [3] mais la notion de décomposition permet d'exprimer des contraintes globales à partir d'autres contraintes globales ou à partir de contraintes d'arité fixe [9]. Nous montrerons que beaucoup de contraintes globales sont soit NP-difficiles à propager (par exemple `Nvalue`, `Sum`), ce qui lie l'existence de propagateurs polynomiaux à la conjecture $P = NP$ [7], soit décomposables en contraintes d'arité fixe préservant la propagation (par exemple `Regular`, `At-most`), ce qui rend l'écriture de propagateurs inutile [2, 5]. Mais nous montrerons aussi que parmi les contraintes globales polynomiales à propager, il y en a qui n'admettent pas de décomposition de taille polynomiale en contraintes d'arité fixe préservant la propagation [8]. La contrainte `alldifferent`, qui admet un propagateur polynomial [12], tombe dans cette catégorie. Ce résultat de non-décomposabilité s'applique aussi aux décompositions en SAT : la propagation unitaire sur un nombre polynomial de clauses ne peut pas simuler la propagation de toutes les contraintes globales polynomiales.

Une alternative possible pour contourner cette limitation serait de définir un ensemble minimal de contraintes globales à implanter dans tout logiciel à contraintes de façon à pouvoir exprimer et propager toutes les autres. Nous montrerons quelques exemples de tentatives pour proposer des contraintes globales de base permettant d'en exprimer beaucoup d'autres [2, 6]. Mais la question de cet ensemble minimal de contraintes reste ouverte.

Les contraintes globales ont récemment été étendues à la programmation par fonctions de coût [11]. Nous verrons que là aussi, les décompositions peuvent permettre dans certains cas de préserver la propagation [1].

Références

[1] D. Allouche, C. Bessiere, P. Boizumault, S. de Givry, P. Gutierrez, S. Loudni, J.P. Meétivier, and T. Schiex. Filtering decomposable global cost functions. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2012)*, Toronto, Canada, 2012.

[2] N. Beldiceanu, M. Carlsson, R. Debruyne, and T. Petit. Reformulation of global constraints based on constraints checkers. *Constraints*, 10(4) :339–362, 2005.

[3] N. Beldiceanu, M. Carlsson, S. Demassey, and T. Petit. Global constraint catalogue : Past, present and future. *Constraints*, 12(1) :21–62, 2007.

[4] C. Bessiere. Constraint propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 3. Elsevier, 2006.

[5] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Slide : A useful special case of the

cardpath constraint. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, pages 475–479, Patras, Greece, 2008.

[6] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Range and roots : Two common patterns for specifying and propagating counting and occurrence constraints. *Artificial Intelligence*, 173(11) :1054–1078, 2009.

[7] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. The complexity of reasoning with global constraints. *Constraints*, 12(2) :239–259, 2007.

[8] C. Bessiere, G. Katsirelos, N. Narodytska, and T. Walsh. Circuit complexity and decompositions of global constraints. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 412–418, Pasadena CA, 2009.

[9] C. Bessiere and P. Van Hentenryck. To be or not to be ... a global constraint. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP'03), LNCS 2833, Springer–Verlag*, pages 789–794, Kinsale, Ireland, 2003.

[10] J.L. Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10 :29–127, 1978.

[11] J.H.M. Lee and K.L. Leung. Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 559–565, Pasadena CA, 2009.

[12] J.C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelth National Conference on Artificial Intelligence (AAAI'94)*, pages 362–367, Seattle WA, 1994.

[13] J.C. Régin. Minimization of the number of breaks in sports scheduling problems using constraint programming. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 57, pages 115–130. 2001.

Problèmes d'optimisation sur des séquences

Thierry Petit[1]

[1] École des Mines de Nantes, LINA, CNRS, INRIA.
Thierry.petit@mines-nantes.fr

Nous nous intéressons aux problèmes d'optimisation impliquant une séquence de variables, qui représentent des coûts. Au travers d'exemples [1] nous montrons que dans de nombreux problèmes des règles métier sont définies sur les variables de la séquence, de façon complémentaire à un critère d'optimisation.

Notre point de vue, confirmé par des expériences, est qu'il peut être nécessaire d'encoder de telles règles par des contraintes et de propager ces contraintes efficacement. En raison de leur contexte d'utilisation, notre objectif a été de concevoir des algorithmes de filtrage ayant une complexité en temps limitée, idéalement linéaire, et n'utilisant pas de structures de données trop lourdes à maintenir. Nous présentons quelques contraintes sur des séquences, leurs contextes d'utilisation, ainsi que les algorithmes de filtrage que nous avons développés :

- Focus [1], qui concentre les valeurs élevées sur un nombre de zones restreint, de tailles limitées. Nous présentons un algorithme de filtrage complet en $O(n)$, où n est le nombre de variables.
- OrderedDistribute [3], une contrainte globale de cardinalité dédiée aux séquences, permettant d'obtenir une répartition équitable des valeurs élevées sur la séquence. Son algorithme de filtrage complet est en $O(\cup_{D_i} + n)$, où $\cup_{D_i}$ est la taille de l'union des domaines.
- SeqBin [2], une contrainte de comptage générique, utile pour représenter plusieurs contraintes telles que Change, Smooth ou IncreasingNvalue. Son algorithme de filtrage est linéaire en la somme des tailles des domaines, $O(\sum_{Di})$.
- IncreasingSum [4], une contrainte de somme dédiée à des séquences de variables dont les valeurs sont croissantes. Nous présentons un algorithme de filtrage aux bornes en $O(n)$.

Nous concluons l'exposé par une discussion sur la résolution de problèmes d'optimisation en programmation par contraintes. D'une part, il existe des techniques de résolution propres à la programmation par contraintes bien que n'étant pas basées sur une recherche systématique. Certaines de ces techniques ont permis de résoudre des problèmes impliquant plusieurs centaines de milliers de variables. D'autre part, la programmation par contraintes est un paradigme robuste à l'ajout de contraintes métier, dont la propagation peut même accélérer le processus de résolution. En d'autres termes, le problème concret, intégrant des règles métier, peut s'avérer aussi voire plus facile à résoudre que le problème "pur". Dans un tel contexte, la possibilité de concevoir un modèle et des algorithmes exploitant la sémantique du problème peut constituer un avantage, qui semble assez spécifique aux techniques de propagation de contraintes.

1. Ordonnancement cumulatif, aide à la composition musicale, ou encore des problèmes d'allocation de ressource.

Références

[1] T. Petit. Focus : A constraint for concentrating high costs. *Research report 12-2-Info, École des Mines de Nantes*, 2012.

[2] T. Petit, N. Beldiceanu, and X. Lorca. A generalized arc-consistency algorithm for a class of counting constraints. *Proc. IJCAI (*revised version Corr/abs/1110.4719*)*, pages 643–648, 2011.

[3] T. Petit and J.-C. Régin. The ordered distribute constraint. *International Journal on Artificial Intelligence Tools*, 20(4) :617–637, 2011.

[4] T. Petit, J.-C. Régin, and N. Beldiceanu. A Θ(n) bound-consistency algorithm for the increasing sum constraint. *Proc. CP*, pages 721–728, 2011.

Approches déclaratives pour l'énumération de motifs intéressants

Lhouari Nourine[1] **Jean-Marc Petit**[2] **Lakhdar Saïs**[2]

[1] LIMOS, Université Blaise Pascal Clermont-Ferrand
[2] LIRIS, Université Claude Bernard Lyon 1
[3] CRIL - CNRS, Université d'Artois
nourine@isima.fr petit@liris.cnrs.fr sais@cril.fr

Ce cours propose de dresser un tour d'horizon des problèmes de découvertes de motifs intéressants dans des masses de données. Après une bréve introduction sur les applications sous-jacentes, nous présenterons leurs principales caractéristiques puis, nous montrerons comment ils peuvent tirer profit de trois champs disciplinaires de l'informatique : l'algorithmique d'énumération, la programmation par contraintes/SAT et les bases de données.

Références

[1] Luc De Raedt and Tias Guns and Siegfried Nijssen Constraint programming for itemset mining. In *Proceedings KDD'2008*, pages 204–212, 2008

[2] Surajit Chaudhuri. Data Mining and Database Systems : Where is the Intersection ? *Data Engineering Bulletin*, 1(21) :4–8, 1998

[3] Vineet Chaoji and Mohammad Al Hasan and Saeed Salem and Mohammed Javeed Zaki. An integrated, generic approach to pattern mining : data mining template library. *Data Min. Knowl. Discov.*, 3(17) :457-495, 2008

[4] Toon Calders and Jef Wijsen. On Monotone Data Mining Languages. In *Proceedings DBPL'2001*, pages 119–132, 2001

[5] F. Flouvat and F. De Marchi and J-M. Petit. ABS : Adaptive Borders Search of frequent itemsets. In *Proceedings FIMI'2004*, 2004

[6] Tomasz Imielinski and Heikki Mannila. A Database Perspective on Knowledge Discovery. *Commun. ACM*, 11(39) :58–64, 1996

[7] Hélène Jaudoin and Frédéric Flouvat and Jean-Marc Petit and Farouk Toumani. Towards a Scalable Query Rewriting Algorithm in Presence of Value Constraints. *J. Data Semantics*, 12 :37–65, 2009

[8] H. Mannila and Hannu Toivonen. Levelwise Search and Borders of Theories in Knowledge Discovery. *DMKD*, 3(1) :241–258, 1997

[9] Siegfried Nijssen and Luc De Raedt IQL : A Proposal for an Inductive Query Language. In *Proceedings KDID'2006*, pages 189–207 2006

[10] Wrobel, Stefan. Inductive logic programming for knowedge discovery in databases. *Relational Data Mining*, 274–99, 2000

[11] Emmanuel Coquery and Saïd Jabbour and Lakhdar Sais. A Constraint Programming Approach for Enumerating Motifs in a Sequence. In *Proceedings ICDM Workshops*, pages 1091–1097 2011

[12] Roberto Bayardo. Efficiently Mining Long Patterns from Databases In *Proceedings ACM SIGMOD*, pages 85–93 1998

[13] M.M. Kanté, A. Mary, V. Limouzy and L. Nourine. Enumeration of Minimal Dominating Sets and Variants. In *Proceedings FCT'2011*, pages 298–309 2011

[14] Alain Gély, Raoul Medina, Lhouari Nourine. About the Enumeration Algorithms of Closed Sets. In *Proceedings ICFCA'2010*, pages 1–16 2010

[15] T. Diallo, N. Novelli, J-M Petit Discovering (frequent) Constant Conditional Functional Dependencies *RIJDMMM*, 2011

Résolution du problème d'allocation de culture par satisfaction de contraintes pondérées

Mahuna Akplogan[1] **Jérome Dury**[2] **Simon de Givry**[1]
Gauthier Quesnel[1] **Alexandre Joannon**[3] **Frédérick Garcia**[1]

[1] INRA, UR875 Biométrie et Intelligence Artificielle
[2] INRA, UMR 1248 AGIR
[3] INRA, UR 980 SAD-Paysage, F-35042 Rennes
{makploga,jdury,degivry,gquesnel,joannon,fgarcia}@toulouse.inra.fr

Résumé

Résoudre un Problème d'Allocation de Cultures (PAC) consiste à planifier, au sein d'une exploitation agricole, l'affectation des cultures aux parcelles de manière à satisfaire un ensemble de contraintes et de préférences agronomiques. Les travaux qui s'adressent à ce problème sont essentiels pour la conception d'outils d'aide à la décision pour la conduite des systèmes de production agricole. De nombreuses méthodes ont été développées en agronomie dans la perspective d'appréhender ce problème. Toutefois, très peu d'entre elles abordent simultanément les dimensions spatiale et temporelle d'un PAC. Dans cet article, nous proposons une approche originale basée sur les CSP pondérés pour appréhender la planification des allocations de cultures. L'utilité des affectations est estimée par une fonction globale combinant linéairement plusieurs critères et qui prends en compte les préférences agronomiques et managériales de l'agriculteur. Nous illustrons les performances de notre approche en nous basant sur une exploitation agricole virtuelle de taille moyenne.

1 Introduction

L'allocation de culture est l'une des décisions les plus importantes auxquelles doit faire face un agriculteur. Elle intervient notamment dans la première phase du processus de production des cultures et permet à l'agriculteur de planifier sur plusieurs années sa stratégie d'occupation du sol. Le terme « allocation de culture » fait référence (i) au choix des cultures à produire, (ii) à la détermination des proportions annuelles de chacune des cultures et (iii) à l'allocation de ces cultures aux parcelles de l'exploitation. Le problème d'allocation de cultures (PAC) qui en découle nécessite la prise en compte d'un ensemble de critères spatiaux (ex : zones cultivables, type de sol) et temporels (ex : contraintes de succession et effets précédents des cultures) interagissant à différents échelles de l'exploitation agricole. Les dimensions spatiale et temporelle d'un PAC sont étroitement liées dans la mesure où le choix des séquences de culture pour chaque parcelle prédétermine l'occupation annuelle des parcelles de l'exploitation agricole.

Dans cet article, nous nous intéressons à la planification de l'allocation des cultures sur un horizon fixe de quelques années. L'utilité des allocations est estimée par une fonction globale combinant linéairement plusieurs critères relatifs aux préférences agronomiques et managériales de l'agriculteur. Contrairement aux différentes approches existantes, notre objectif est de proposer une nouvelle direction capable d'appréhender l'ensemble des facteurs spatiaux et temporels exploités par l'agriculteur. Ces critères sont formalisés dans le cadre des CSP pondérés sous la forme de contraintes dures et de préférences de l'agriculteur. Notre choix des contraintes repose sur une étude réalisée par Dury [5] sur les différentes pratiques des agriculteurs. En raison de la multiplicité des critères pris en compte lors de la résolution d'un PAC, notre approche semble être une voie prometteuse. Ce travail préliminaire à de plus été exploité pour la mise en œuvre d'un outil (CRASH - *Crop Rotation and Allocation Simulator using Heuristics* -) d'aide à l'allocation explicite des cultures.

L'article est organisé comme suit. Dans la section 2, nous décrivons le problème d'allocation de cultures en introduisant quelques définitions spécifiques. La sec-

tion 3 présente les approches existantes et souligne les limites de celles-ci. Dans la section 4, nous introduisons notre formalisation des contraintes dans le cadre des CSP pondérés. Ensuite, la section 5 illustre les performances de l'approche proposée en se basant sur une exploitation agricole virtuelle. Enfin, dans la section 6, nous discutons de la pertinence et des limites de notre proposition.

2 Problème d'allocation de culture (PAC)

2.1 Description globale du problème

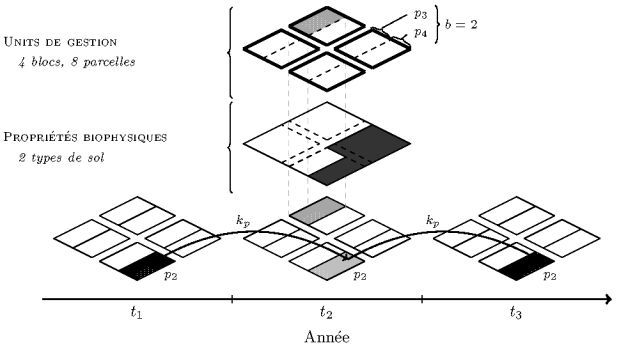


Fig. 1 – Représentation schématique des concepts spatio-temporels d'un PAC (t_i : année, b : bloc, p_j : parcelle, k_p : effet précédent)

Le problème d'allocation de cultures (PAC) est un problème de planification sur un horizon fini $\mathcal{H}$, durant lequel des cultures sont affectées aux parcelles (Figure 1) de manière à prendre en compte un ensemble de concepts agronomiques relatifs (i) à l'organisation spatiale de l'exploitation et (ii) à la succession temporelle des cultures sur les parcelles.

La dimension spatiale repose sur différents niveaux d'organisation appelés *unités de gestion* (Figure 1). Ces unités de gestion sont décidées par l'agriculteur et déterminent la gestion du travail quotidien et l'allocation des ressources. Afin de simplifier notre exemple, nous ne considérons que les deux principales unités de gestion qui sont : les *parcelles* et les *blocs*. Le premier se rapporte à la gestion annuelle des cultures. Une parcelle est une entité spatiale continue d'une exploitation agricole qui est homogène en terme de la culture produite une année donnée [14]. Les délimitations des parcelles sont ajustées d'une année à l'autre. Comme l'indique la Figure 1, les *blocs* sont des sous ensembles de parcelles gérés de la même manière par l'agriculteur. Un bloc caractérise un système de culture c'est-à-dire, une séquence de culture et l'utilisation du même itinéraire techniques pour la conduite de ses cultures (ex : irrigation, fertilisation). Contrairement aux parcelles, nous considérons ici que la délimitation des blocs est fixe. Cette délimitation dépend cependant des caractéristiques de l'exploitation telles que la disponibilité des ressources (ex : accès à une source d'irrigation) ou encore les propriétés biophysiques (ex : type de sol, topologie, accessibilité des parcelles). Ces *propriétés biophysiques* sont également utilisées afin de déterminer si une culture peut être produite dans de bonnes conditions sur un type de sol donné.

Au niveau temporel, certaines séquences de culture sont interdites ou déconseillées en raison de leurs impacts sur la fertilité du sol, l'apparition de pathogènes et la prépondérance de germes de mauvaises herbes sur les parcelles. La qualité d'une séquence de culture est évaluée suivant deux indicateurs qui sont : le *délai de retour minimum* d'une culture v ($rt(v)$) et *l'effet précédent* (k_p). Le *délai de retour minimum* définit le nombre d'années entre deux instances successives d'une même culture sur une même parcelle. Par exemple, dans la Figure 1, le délai de retour minimum de la culture produite sur p_2 (bloc 3) à la date t_1 est égale à 2 ans. Plus généralement, soient t et t' deux différentes années ($t < t'$), p_j une parcelle et v une culture, $p_j(t) = p_j(t') = v$ **si** $(t' - t) \geq rt(v)$.

L'effet précédent (k_p) détermine la variation de l'état du sol. Il est représenté dans notre cas par un indicateur k_p [12] qui mesure le bénéfice ou le déficit d'une culture sur son suivant. En nous basant sur le k_p, nous pouvons mesurer l'utilité d'une séquence de culture par rapport à une autre. Par ailleurs, pour certains auteurs comme [4], la stabilité d'un système de culture ne peut être garantie que lorsque les séquences de cultures sur les parcelles sont répétables dans le temps. Pour ce faire, les séquences de culture que nous recherchons doivent être répétables tout en respectant les délais de retour des cultures. Cette notion connue sous le nom de « *rotation de culture* » est couramment utilisée par les agriculteurs.

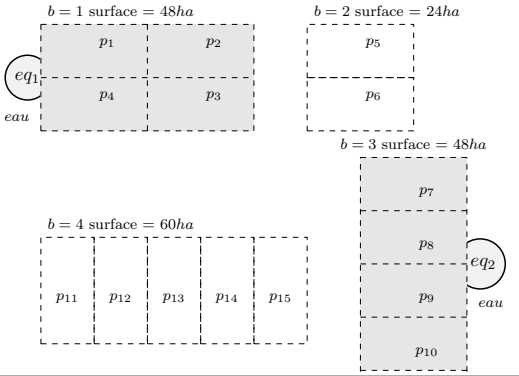
2.2 Description des contraintes

Résoudre un problème d'allocation de culture (PAC) consiste à affecter sur un horizon fixe $\mathcal{H}$ des cultures aux parcelles. Chaque affectation des cultures doit satisfaire un ensemble de contraintes dures et de préférences.

Les contraintes dures que nous retenons dans le cadre de ces travaux sont notamment les *délais de retour minimum* des cultures, l'*historique* des parcelles, les propriétés *physiques* (types de sol, accessibilité des ressources) des parcelles. Les préférences sont quant à elles relatives aux *effets précédents* (k_p) et à l'équilibre spatial et temporel des proportions de culture et ceci sous contrainte de ressources. Les contraintes dures et les préférences sont définies aussi bien au niveau :

– *des parcelles* afin d'exprimer pour chacune d'entre elles la possibilité (voir l'impossibilité) d'un redécoupage ou d'une fusion,
– *des blocs* afin d'exprimer la compatibilité spatiale des cultures aux parcelles, les délais de retour minimum et les effets précédents des cultures,
– *de l'exploitation* afin d'exprimer les objectifs de production ou l'usage des ressources.

Considérons le problème d'allocation de culture décrit à la Figure 2. Ce problème, décrit une exploitation virtuelle dont la superficie totale est de 180 ha. L'exploitation ainsi considérée comporte 4 blocs ($b \in \{1, 2, 3, 4\}$) et 15 parcelles $p_{[1-15]}$ de 12 ha chacune. L'exploitation virtuelle ainsi considérée correspond à un problème réel de taille moyenne. Quatre différentes cultures sont cultivées sur l'exploitation. Il s'agit du *blé d'hiver* (BH), de *l'orge de printemps* (OP), du *maïs* (MA) et du *colza d'hiver* (CH). Le maïs est la seule culture irriguée de l'exploitation. Comme l'indique la Figure 2, seuls les blocs 1 (eq_1) et 3 (eq_2) sont équipés en matériels d'irrigation fixes. Le quota d'eau annuel destiné à l'irrigation est de $10000m^3$ dont $6000m^3$ pour eq_1 et $4000m^3$ pour eq_2. On distingue deux types de sols sur l'ensemble de l'exploitation. Le premier type de sol est celui des blocs 1 et 3 tandis que le second est celui des blocs 2 et 4. Les allocations réalisées par l'agriculteur durant les cinq dernières années sont décrites par le tableau de la Figure 2.



parcelle / année	t1	t2	t3	t4	t5
$\mathbf{p}_1$	MA	MA	BH	OP	MA
$\mathbf{p}_2$	OP	MA	MA	BH	OP
$\mathbf{p}_3$	BH	OP	MA	MA	BH
$\mathbf{p}_4$	MA	BH	OP	MA	MA
$\mathbf{p}_5$	BH	OP	BH	CH	BH
$\mathbf{p}_6$	OP	BH	CH	BH	OP
$\mathbf{p}_7$	MA	MA	MA	MA	MA
$\mathbf{p}_8$	MA	MA	MA	MA	MA
$\mathbf{p}_9$	MA	MA	MA	MA	MA
$\mathbf{p}_{10}$	MA	MA	MA	MA	MA
$\mathbf{p}_{11}$	BH	CH	BH	OP	BH
$\mathbf{p}_{12}$	CH	BH	OP	BH	CH
$\mathbf{p}_{13}$	BH	OP	BH	CH	BH
$\mathbf{p}_{14}$	OP	BH	CH	BH	OP
$\mathbf{p}_{15}$	BH	CH	BH	OP	BH

FIG. 2 – Exploitation virtuelle de 4 blocs, 15 parcelles de 12ha chacune. Les parcelles grises ont un accès à l'eau (eq_1, eq_2). Le tableau contient l'historique des parcelles pour les cinq dernières années.

2.2.1 Définition des contraintes dures

1. **h-SCC** - *compatibilité spatiale des cultures* : le colza d'hiver (CH) ne peut pas être affecté aux parcelles élémentaires dont le type de sol est 1 (blocs $1, 3$).
2. **h-EQU** - *égalité des parcelles* : les parcelles p_7 (respectivement p_9) et p_8 (respectivement p_{10}) doivent être affectées à la même culture. Ces parcelles ont été choisies par l'agriculteur pour être conduites de la même manière.
3. **h-HST** - *Historique des parcelles* : pour les cinq dernières années, l'occupation du sol est fixée et présentée dans le tableau de la Figure 2.
4. **h-TSC** - *Délai de retour minimum* : pour chaque couple parcelle/culture, les délais de retour minimum suivants doivent être respectés : $rt(BH) = 2$, $rt(OP) = 3$, $rt(MA) = 2$ et $rt(CH) = 3$.
5. **h-CCS** - *Rotation culturale* : pour chacune des parcelles, la séquence de culture après l'historique doit être indéfiniment répétables sans enfreindre la contrainte de délai de retour minimum.
6. **h-RSC** - *Capacité de ressources* : l'exploitation dispose de quantités fixes de ressources. Pour chacune d'entre elles, le cumul des quantités nécessaires à la production des cultures affectées doit être inférieur à la capacité disponible. Par exemple, seul le maïs est irrigable. Sachant qu'il faut $165m^3$ d'eau par hectare de maïs, sa production annuelle sur le bloc 1 ne peut excéder 36.36 ha.
7. **h-SCA** - *Collection de cultures par bloc* : sur l'horizon de planification, le même sous ensemble de cultures doit être assigné à l'ensemble des parcelles d'un bloc.

2.2.2 Définition des préférences

1. **s-TOP** - *Topologie de l'exploitation* : les parcelles assignées à une même culture doivent être spatialement groupées. Autrement dit, pour limiter la perte de temps liée aux déplacements entre parcelles durant la conduite journalière des cultures, il est préférable de grouper autant que possible les cultures. En conséquence, chaque parcelle isolée

	culture précédente			
	BH	OP	MA	CH
BH	4	1	1	0
OP	2	3	1	0
MA	0	0	3	0
CH	0	0	0	4

FIG. 3 – Table des effets précédents (k_p)

sera pénalisée par un coût δ_2.

2. **s-SBC** - *Équilibre spatial des cultures* : pour certaines cultures, les objectifs de production annuelle sont prédéfinis. Par exemple, dans le PAC ci-dessus considéré, les proportions annuelles de maïs (MA) doivent être comprises entre $[24, 48]$ *ha* sur le bloc 1 et $[12, 24]$ *ha* sur le bloc 3. Toutes les déviations seront pénalisées par un coût δ_1.
3. **s-TBC** - *Équilibre temporel des cultures* : pour de raisons purement agronomiques, les objectifs de production de certaines cultures sur certaines parcelles sont prédéfinis. Par exemple, la proportion de colza d'hiver (CH) produite sur chaque parcelle doit être comprise entre $[12, 24]$ *ha*. Toutes les déviations seront pénalisées par un coût δ_3.
4. **s-CSQ** - *Effet précédent* : chaque succession de culture est associée à un coût k_p qui mesure l'effet précédent. La Figure 3 présente les valeurs de k_p.

En pratique, les coûts k_p, δ_1, δ_2 et δ_3 sont définis tels que $\sum k_p \gg \sum \delta_1 \gg \sum \delta_2 \gg \sum \delta_3$. Ce faisant, nous hiérarchisons de manière réaliste les préférences de l'agriculteur. En effet, les effets précédents k_p doivent être avant tout minimisés en raison des conséquences qu'ils peuvent engendrer sur les cultures suivantes. L'équilibre spatial des cultures (coût δ_1) définit implicitement les revenus annuels de l'agriculteur. De ce fait, il doit être garanti du mieux possible. Ensuite, la charge de travail doit être réduite en groupant (coût δ_2) les cultures identiques. Enfin, l'équilibre temporel des cultures (δ_3) doit être respecté.

3 Analyse de l'existant

Depuis Heady [7], différentes approches ont été proposées pour la recherche de la meilleure affectation de cultures aux parcelles [11]. La plupart des travaux sont basés soit sur l'optimisation de la marge brute [1] des cultures soit sur l'utilisation de filtres agronomiques [2] (ex : règles de succession, quantités de produits phytosanitaires dans le sol, etc.) pour fournir des successions de cultures admissibles. Dans les différentes approches existantes, le PAC a été abordé soit sous la forme d'une optimisation de proportions annuelles des cultures [10, 17] soit sous la forme d'une recherche des successions (rotations) de cultures admissibles [6, 4]. Ces deux aspects constituent cependant deux dimensions (spatiale et temporelle) du PAC. De plus, à défaut d'être spatialement explicites, les solutions proposées sont des agrégations de propositions de cultures sur un ensemble de parcelles de différents types. Nous proposons d'une part, de prendre en compte simultanément les deux dimensions du problème et d'autre part, de rendre les solutions proposées spatialement explicites.

4 Modèle CSP pondérés d'un PAC

4.1 Le cadre des CSP pondérés

Le formalisme des réseaux de contraintes (CSP) ne permet pas de prendre en compte facilement les préférences de l'agriculteur. Nous nous basons ici sur les réseaux de contraintes pondérées (*Weighted CSP* - WCSP) qui nous semblent plus appropriés à la résolution des problèmes d'optimisation. Le cadre WCSP [13] est une extension qui ajoute aux CSP une structure de valuation permettant de définir une structure algébrique caractérisant les coûts associés à certaines combinaisons de valeurs. Un WCSP est défini par un triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{W} \rangle$ avec :

- $\mathcal{X} = \{x_1, \cdots, x_n\}$ un ensemble fini de variables.
- $\mathcal{D} = \{D_1, \cdots, D_n\}$ un ensemble fini de domaines de variables. Chaque variable $x_i \in \mathcal{X}$ est associée à un domaine fini de valeur $D_i \in \mathcal{D}$.
- $\mathcal{W} = \{W_{S_1}, \cdots, W_{S_e}\}$ un ensemble de fonctions de coûts. Soit $l(S_i)$ l'ensemble des combinaisons de valeurs sur la portée S_i. Chaque fonction de coûts $W_{S_i} \in \mathcal{W}$ est définie par $W_{S_i} : l(S_i) \rightarrow [0, m]$ avec $m \in [1, \cdots, +\infty]$.

La solution d'un WCSP est une affection complète $A \in l(\mathcal{X})$ qui minimise $\sum_{\mathcal{W}_{S_i} \in \mathcal{W}} \mathcal{W}_{S_i}(A[S_i])$ où $A[S_i]$ est la projection d'une affectation de valeurs sur le sous ensemble de variables S_i.

4.2 Définition du problème d'allocation de culture

4.2.1 Notion de parcelle élémentaire

Afin de prendre en compte les différentes contraintes numériques, nous proposons d'échantillonner l'exploitation agricole en **parcelles élémentaires**. Chaque parcelle élémentaire est définie comme *une entité spatiale homogène, indivisible, ayant le même historique et les mêmes propriétés biophysique*. Toutes les parcelles élémentaires sont de tailles identiques. Par exemple en divisant chacune des parcelles (Figure 2) en 8 parcelles élémentaires de 1.5 *ha*. Le nombre total de parcelles élémentaires serait égal à 120. Dans

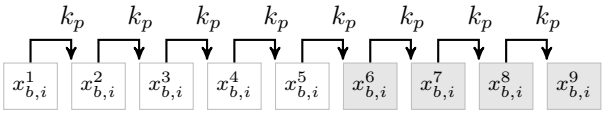


FIG. 4 – Séquence de variables définissant la parcelle élémentaire i du bloc b.

ce cas, les parcelles ne seront que des combinaisons de plusieurs parcelles élémentaires.

4.2.2 Définition du problème

Le PAC est défini sur un horizon fini $\mathcal{H}$ par un ensemble de parcelles élémentaires et de cultures. Nous définissons le WCSP associé de manière suivante.

- $\mathcal{X}$ un ensemble de variables $x^t_{b,i} \in \mathcal{X}$. Chaque variable $x^t_{b,i}$ définit la parcelle élémentaire i du bloc b, $i \in \mathcal{N}_b$, $b \in [1, \mathcal{B}]$ ($\mathcal{B} = 4$ et $\mathcal{N}_1 = [1, 32]$ dans le PAC décrit par la Figure 2) à la date t ($t \in [1, \mathcal{H}]$). Ainsi, chaque parcelle élémentaire est décrite par $\mathcal{H}$ variables correspondant à l'occupation de la parcelle élémentaire à chaque instant. Soient $[1, h]$ et $[h+1, \mathcal{H}]$ respectivement les instants du passé (historique) et du futur. Pour $\mathcal{H} = 9$ et $h = 5$, la Figure 4 représente les 9 variables qui définissent la parcelle élémentaire i du bloc b. Les cinq premières (sommets blancs) sont les variables d'historique.
- $\mathcal{D}$ les domaines $D_{b,i}$ des variables $x^t_{b,i}$ est l'ensemble des cultures possibles sur toutes les parcelles élémentaires. Considérant le PAC de la Figure 2, $\forall b \in [1, \mathcal{B}], \forall i \in \mathcal{N}_b, \quad D_{b,i} = \{1, 2, 3, 4\} = \{BH, OP, MA, CH\}$.
- $\mathcal{W}$ les fonctions de coûts $\mathcal{W}$ se divisent en cinq différents : (1) fonctions de coût tabulaires (arité maximale de 5), (2) des contraintes globales same, (3) des contraintes globales regular, (4) des contraintes globales de cardinalité gcc, (5) des contraintes globales soft-gcc. Ces fonctions de coût sont détaillées dans les sections suivantes.

4.3 Fonctions de coût simples

Les contraintes dures et les préférences h-SCC, h-EQU, h-HST, s-TOP et s-CSQ sont définies par des fonctions de coût tabulaires.

h-SCC : $\forall t \in [h+1, \mathcal{H}]$, $\forall b \in \mathcal{B}$, $\forall i \in \mathcal{N}_b$, soit $W^{SCC}_{x^t_{b,i}}$ une fonction de coût unaire associée à la compatibilité spatiale des cultures :

$\forall a \in D_{b,i}$

$$W^{SCC}_{x^t_{b,i}}(a) = \begin{cases} \infty & \textit{si } a \textit{ est interdit sur la parcelle élémentaire } i \textit{ du bloc } b \\ 0 & \textit{sinon} \end{cases} \quad (1)$$

h-EQU : $\forall t \in [h+1, \mathcal{H}]$, $\forall b \in \mathcal{B}$, pour tous les couples de parcelles élémentaires $(i, j) \in \mathcal{N}_b \times \mathcal{N}_b$ qui doivent être gérés de la même manière, on définit une contrainte d'égalité $W^{EQU}_{x^t_{b,i}, x^t_{b,j}}$ sur les variables $x^t_{b,i}$ et $x^t_{b,j}$.

$\forall a \in D_{b,i}, \forall a' \in D_{b,j}$

$$W^{EQU}_{x^t_{b,i}, x^t_{b,j}}(a, a') = \begin{cases} 0 & si\ a = a' \\ \infty & sinon \end{cases} \quad (2)$$

h-HST : $\forall b \in \mathcal{B}$, $\forall i \in \mathcal{N}_b$, $\forall t \in [1, h]$, soit $W^{HST}_{x^t_{b,i}}$ une fonction de coût unaire associée à une variable d'historique d'une parcelle élémentaire.

$\forall a \in D_{b,i}$

$$W^{HST}_{x^t_{b,i}}(a) \quad = \quad \begin{cases} 0 & \text{si } a = \mathsf{historic}(x^t_{b,i}) \\ \infty & sinon \end{cases} \quad (3)$$

où $\mathsf{historic}(x^t_{b,i})$ retourne la valeur de la parcelle élémentaire i du bloc b à la date t.

s-TOP : $\forall t \in [1, \mathcal{H}]$, $\forall b \in \mathcal{B}$, $\forall i \in \mathcal{N}_b$, soit W^{TOP}_S une fonction de coût n-aire associée à la topologie de l'exploitation. Nous définissons une fonction de voisinage $\mathsf{neighbor}(i)$ qui retourne les parcelles élémentaires $j \in \mathcal{N}_b$ ayant au moins une frontière commune avec i. Dans notre exploitation virtuelle nous considérons les 4 plus proches voisins suivant la notion de voisinage de von Neumann. Ainsi, la portée S est égale à $\{x^t_{b,i}, x^t_{b,n}, x^t_{b,s}, x^t_{b,e}, x^t_{b,w}\}$ où les parcelles élémentaires n, s, e, o sont les 4 voisins respectivement au nord, sud, est et ouest de i. $\forall a \in D_{b,i}, \forall a_n \in D_{b,n}, \forall a_s \in D_{b,s}, \forall a_e \in D_{b,e}, \forall a_w \in D_{b,w}$

$$W^{TOP}_S(a, a_n, a_s, a_e, a_w) = \begin{cases} 0 & si\ a = a_n = a_s \\ & \quad = a_e = a_w \\ \delta_2 & sinon \end{cases} \quad (4)$$

En fonction de la position de i dans le bloc, l'arité de W^{TOP}_S pourrait se réduire à 3 ou 4 (coin et extrémité).

s-CSQ : $\forall t \in [1, \mathcal{H}]$, $\forall b \in \mathcal{B}$, $\forall i \in \mathcal{N}_b$, soit $W^{CSQ}_{x^t_{b,i}, x^{t+1}_{b,i}}$ une fonction de coût binaire associée à l'effet précédent k_p. Nous définissons une fonction $\mathsf{KP}(a, a')$ qui retourne le coût k_p de la succession de culture a' précédé de a :

$\forall a \in D_{b,i}, \forall a' \in D_{b,i}$

$$W^{CSQ}_{x^t_{b,i}, x^{t+1}_{b,i}}(a, a') \quad = \quad \mathsf{KP}(a, a') \quad (5)$$

4.4 Collection de cultures par bloc via la contrainte « same »

h-SCA : considérons un bloc b, le sous ensemble de $(\mathcal{H}-h)*|\mathcal{N}_b|$ variables du futur $x^t_{b,i}$ (avec $t \in [h+1, \mathcal{H}]$)

associé à chaque parcelle élémentaire i dans b doit être assignée à une seule et même collection de culture. Ainsi, $\forall (i,j) \in \mathcal{N}_b \times \mathcal{N}_b$ (avec $i \neq j$), l'ensemble des valeurs affectées à la séquence de variables définissant i est une permutation de celui de j. Nous proposons de modéliser h-SCA par une contrainte globale same [3]. Pour chaque bloc, on choisit une parcelle élémentaire de *référence* i. On définit ensuite une fonction de coût W_S^{SCA} d'arité $2 * (\mathcal{H} - h)$ associée à chaque pair de séquences de variables qui définissent $x_{b,i}^t$ et $x_{b,j}^t$ $(i \neq j)$. La portée $S = \{x_{b,i}^{h+1}, \cdots, x_{b,i}^{\mathcal{H}}, x_{b,j}^{h+1}, \cdots, x_{b,j}^{\mathcal{H}}\}$. Soient $A[x_{b,i}^{h+1}, \cdots, x_{b,i}^{\mathcal{H}}]$ et $A[x_{b,j}^{h+1}, \cdots, x_{b,j}^{\mathcal{H}}]$ les deux sous affectations des variables de S. La contrainte W_S^{SCA} impose que $A[x_{b,i}^{h+1}, \cdots, x_{b,i}^{\mathcal{H}}]$ soit une permutation de $A[x_{b,j}^{h+1}, \cdots, x_{b,j}^{\mathcal{H}}]$.

$$W_S^{SCA} = \mathsf{same}(\underbrace{x_{b,i}^{h+1}, \cdots, x_{b,i}^{\mathcal{H}}}_{i}, \underbrace{x_{b,j}^{h+1}, \cdots, x_{b,j}^{\mathcal{H}}}_{j}) \quad (6)$$

4.5 Séquence de culture via la contrainte « regular »

Les contraintes h-TSC et h-CCS portent sur la succession temporelle des cultures. Nous proposons de les modéliser en utilisant des contraintes globales regular [15]. Ainsi, $\forall t \in [1, \mathcal{H}]$, $\forall b \in \mathcal{B}$, $\forall i \in \mathcal{N}_b$, $\forall a \in D_{b,i}$, soient $M_{b,i}^a$ un automate non déterministe à états finis (NFA), $\mathcal{L}(M_{b,i}^a)$, $\mathcal{L}(M_{b,i}^a)$ le langage définit par l'automate et $S_{b,i}$ la séquence de $\mathcal{H}$ variables décrivant une parcelle élémentaire i du bloc b. Une solution de la contrainte $\mathsf{regular}(S_{b,i}, M_{b,i}^a)$ est une affectation $A[S_{b,i}]$ telle que $A[S_{b,i}] \in \mathcal{L}(M_{b,i}^a)$.

h-TSC : pour chaque parcelle élémentaire $x_{b,i}$, nous définissons pour chacune des cultures $a \in D_{b,i}$ un langage $\mathcal{L}(M_{b,i}^a)$ qui décrit le délai de retour de a. Autrement dit, $(x_{b,i}^t = x_{b,i}^{t'})$ *ssi* $x_{b,i}^{t'}$ $\forall t' \neq t$, $t' \geq t + rt(a)$. Nous définissons un $\mathsf{regular}(S_{b,i}, M_{b,i}^a)$. Par exemple, pour le cas où $a = CH$, le délai de retour $rt(CH) = 3$ ans. L'automate $M_{b,i}^a$ est décrit comme sur la Figure 5. Dans cet exemple, l'état initial est 0 et les états terminaux sont $5, 6, 7$. Les arcs sont étiquetés avec les valeurs des cultures.

Comme l'indique le NFA de la Figure 5, les variables d'historique sont exploitées afin d'imposer le délai de retour sur les variables du futur. Pour ce faire, nous définissons pour chaque parcelle élémentaire, une fonction de coût $W_{S_{b,i}}^{TSC^a}$ d'arité $\mathcal{H}$ telle que :

$\forall b \in \mathcal{B}, \forall i \in \mathcal{N}_b, \forall a \in D_{b,i}$

$$W_{S_{b,i}}^{TSC^a} = \mathsf{regular}(x_{b,i}^1, \cdots, x_{b,i}^t, \cdots, x_{b,i}^{\mathcal{H}}, M_{b,i}^a) \quad (7)$$

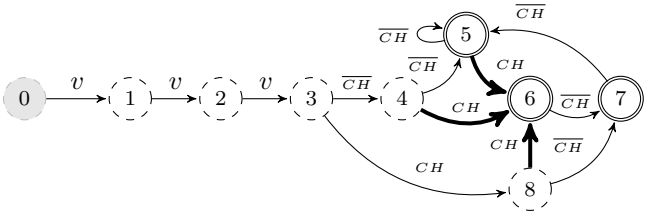


FIG. 5 – Automate pour la culture $a = CH$ avec $(rt(CH) = 3$ et $h = 5)$. v désigne toutes les valeurs de $D_{b,i}$. La notation $\overline{CH}$ correspond à $D_{b,i} \setminus \{CH\}$. Le langage associé accepte toutes les séquences sur $\mathcal{H}$ pour lesquelles le délai de retour est respecté pour l'ensemble des variables du futur (ex : CH-OP-CH-OP-CH-BH-OP-CH-BH).

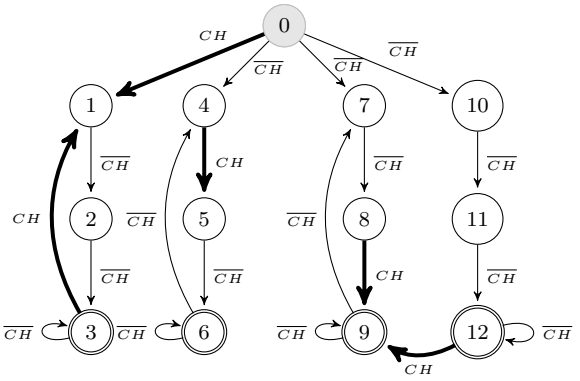


FIG. 6 – Automate cyclique pou les cultures $a = CH$ avec $rt(CH) = 3$ et $\mathcal{H} - h = 4$.

h-CCS : pour chaque parcelle élémentaire $x_{b,i}$, nous combinons h-TSC avec la contrainte de rotation culturale définie également par une contrainte globale regular. Cette contrainte h-CCS assure que la séquence de culture après l'historique est indéfiniment répétable sans violation de la contrainte de délai de retour. La Figure 6 décrit un NFA cyclique pour la culture CH. L'état initial est 0. Les états terminaux sont $3, 6, 9, 12$. La portée de la fonction de coût se réduit aux variables du futur.

$\forall b \in \mathcal{B}, \forall i \in \mathcal{N}_b, \forall a \in D_{b,i}$

$$W_{S_{b,i}}^{CCS^a} = \mathsf{regular}(x_{b,i}^{h+1}, \cdots, x_{b,i}^{\mathcal{H}}, M_{b,i}^a) \quad (8)$$

4.6 Capacité de ressources via la contrainte globale de cardinalité

Dans un PAC, chaque parcelle élémentaire consomme une quantité fixe de ressources en fonction de ses exigences qualitatives (type de culture) ou numériques (superficie de la parcelle élémentaire, dose d'irrigation). Par exemple, contrairement au blé d'hiver, le maïs est une culture irriguée. La gestion

des ressources est un problème de séquencement et de comptage des quantités allouées. L'approche de résolution classique est basée sur la recherche du plus court chemin sous contrainte de ressource [9]. Considérant les hypothèses 1 et 2, nous suggérons de réduire le problème d'allocation de ressources à un problème de comptage.

Hypothèse 1 *: Les ressources sont supposées consommables et systématiquement renouvelables chaque année sans aucune fonction de production (ex : un quota annuel d'eau pour l'irrigation).*

Cette hypothèse est très proche de la réalité car les agriculteurs disposent généralement d'un quota annuel d'eau. Il en est de même pour le nombre d'heures de travail annuel disponible en fonction de la législation.

Hypothèse 2 *:* $\forall t \in [1, \mathcal{H}]$, $\forall (b, b') \in \mathcal{B} \times \mathcal{B}$ *un couple de blocs,* $\forall (i, j) \in \mathcal{N}_b \times \mathcal{N}'_b$ *un couple de parcelles élémentaires. Les superficies des parcelles élémentaires* i *et* j *sont considérées comme étant équivalentes en fonction de la taille du problème.*

Sous ces hypothèses, l'allocation annuelle des ressources est vue comme un problème de comptage à chaque instant $t \in [h+1, \mathcal{H}]$. Ainsi, connaissant la capacité annuelle des ressources, nous définissons pour chaque instant $t \in [h+1, \mathcal{H}]$ une borne supérieure et une borne inférieure du nombre de variables $x^t_{i,b}$ qui peuvent être affectées à une culture donnée et ceci en fonction des exigences qualitatives et numériques de l'exploitation.

h-RSC : la contrainte de capacité de ressource h-RSC est modélisée par une contrainte globale de cardinalité gcc [16] portant sur les affectations de cultures aux parcelles élémentaires.

$\forall t \in [h+1, \mathcal{H}]$, soit $W^{RSC}_{S^t_b}$ une contrainte globale d'arité $|\mathcal{N}_b|$ associée aux capacités de ressources.

Sachant $S^t_b = (x^t_{b,1}, \cdots, x^t_{b,|\mathcal{N}_b|})$, la contrainte globale de cardinalité (gcc) spécifie, pour chaque valeur $a \in \bigcup D_{b,i}$, une borne inférieure $lb(a)$ et une borne supérieure $ub(a)$ correspondant au nombre d'occurrence de la valeur a dans l'affectation $A[S^t_b]$.

$$W^{RSC}_{S^t_b} \quad = \quad \mathsf{gcc}(S^t_b, lb, ub) \tag{9}$$

admet une solution s'il existe une affectation de S^t_b telle que :

$$\forall a \in \bigcup D_{b,i},\ lb(a) \leq |\{x^t_{b,i} \in S^t_b | x^t_{b,i} = a\}| \leq ub(a) \tag{10}$$

Par exemple, considérons le bloc $b = 1$ de l'exploitation virtuelle (Figure 2) et supposons un échantillonnage de l'espace en parcelles élémentaires de 1.5 ha (section 4.2.1). Sachant qu'il faut $165m^3$ d'eau par hectare de maïs, les superficies minimum et maximum de maïs sur ce bloc sont 0 et 36.36 ha. Ainsi, sur le bloc $b = 1$, la borne inférieure du nombre d'occurrence de maïs est $lb(MA) = 8 \times \lfloor \frac{0\ ha}{12\ ha} \rfloor = 0$. La borne supérieure $ub(MA) = 8 \times \lfloor \frac{36.36\ ha}{12\ ha} \rfloor = 24$.

4.7 Équilibre spatial et temporel des cultures via la contrainte globale « soft-gcc »

Les préférences relatives aux proportions annuelles (s-SBC) et pluriannuelles des cultures (s-TBC) sont définies par des contraintes globales (soft-gcc). Elles autorisent la relaxation du gcc associé dans le cas où le problème est sur-contraint.

Considérons un soft-gcc(S, lb, ub, δ). Nous définissons pour tout $a \in \bigcup D_{b,i}$.

$$ovf(S, a) = max(|\{x^t_{b,i} | x^t_{b,i} = a\}| - ub(a), 0)$$
$$unf(S, a) = max(lb(a) - |\{x^t_{b,i} | x^t_{b,i} = a\}|, 0)$$

où lb et ub sont respectivement la borne inférieure et la borne supérieure de chacune des cultures, δ une variable de coût appartenant à un domaine fini. Nous utilisons la mesure de violation orientée variables μ [8] qui se base sur le nombre minimum de variables dont les valeurs doivent être changées afin de satisfaire la contrainte gcc associée. Si $\sum_{a \in \bigcup D_{b,i}} lb(a) \leq |S| \leq \sum_{a \in \bigcup D_{b,i}} ub(a)$, la mesure de violation μ est exprimé par :

$$\mu(S) = max\left(\sum_{a \in \bigcup D_{b,i}} ovf(S, a), \sum_{a \in \bigcup D_{b,i}} unf(S, a) \right) \tag{11}$$

La fonction de coût associée à chaque contrainte soft-gcc(S, lb, ub, δ) est $W = \mu(S) \times \delta$. Sur la base de cette définition les contraintes s-SBC et s-TBC sont formalisées comme ci-dessous.

s-SBC : $\forall t \in [h+1, \mathcal{H}]$, $\forall b \in \mathcal{B}' \subseteq \mathcal{B}$. Soit $W^{SBC}_{S^t_b}$ une contrainte soft-gcc d'arité $|\mathcal{B}'|$ associée au bloc b à la date t. La portée $S^t_b = \{x^t_{b,i} | i \in \mathcal{N}_b\}$.

$$W^{SBC}_{S^t_b} \quad = \quad \mathsf{soft\text{-}gcc}(S^t_b, lb, ub, \delta = \delta_2) \tag{12}$$

s-TBC : $\forall b \in \mathcal{B}' \subseteq \mathcal{B}$, $\forall i \in \mathcal{N}_b$. Soit $W^{TBC}_{S_{b,i}}$ une contrainte soft-gcc d'arité $(\mathcal{H} - h)$ associée à chaque parcelle élémentaire i. La portée $S_{b,i} = \{x^{h+1}_{b,i}, \cdots, x^{\mathcal{H}}_{b,i}\}$. À l'exception de la portée et de la variable de coût, $\delta = \delta_3$, $W^{TBC}_{S_{b,i}}$ est définie exactement comme $W^{SBC}_{S^t_b}$.

Par exemple, considérons la préférence s-TBC décrite dans la section 2.2.2. La proportion de colza

d'hiver (CH) produite sur chaque parcelle doit être comprise entre $[12, 24]$ ha. Pour chaque séquence de variables $\{x_{b,i}^{h+1}, \cdots, x_{b,i}^{\mathcal{H}}\}$ qui définit une parcelle élémentaire, la borne inférieure du nombre d'occurrence de colza d'hiver est $lb(CH) = 8 \times \lfloor \frac{12\ ha}{12\ ha} \rfloor = 8$. La borne supérieure est $ub(CH) = 8 \times \lfloor \frac{24\ ha}{12\ ha} \rfloor = 16$.

5 Implémentation

5.1 Description des instances de PAC

Nous avons conduit nos expérimentations en utilisant quatre instances de l'exploitation virtuelle décrite à la Figure 2. Chaque instance correspondant à un échantillonnage spécifique de l'exploitation en parcelles élémentaires. Nous avons progressivement augmenté le nombre total de parcelles élémentaires de 15 à 120 (15, 30, 60, 120).

Pour l'instance comportant 15 parcelles élémentaires, $|\mathcal{N}_1| = |\mathcal{N}_3| = 4, |\mathcal{N}_2| = 2$ et $|\mathcal{N}_4| = 5$ avec $|\mathcal{N}_i|$ le nombre de parcelles élémentaires dans le bloc i. Dans cette instance du problème, les parcelles de 12 ha (Figure 2) sont supposées élémentaires. Partant de cette instance nous découpons successivement chaque parcelle élémentaire de 12ha en 2, 4 puis 8 plus petites parcelles élémentaires. Ces échantillonnages sont représentatifs de différentes tailles d'exploitations. L'horizon de planification $\mathcal{H}$ est égal à 9 années. Les cinq premières années sont associées à l'historique présenté dans le tableau de la Figure 2. Les quatre dernières années sont associées au futur.

Notons que dans l'exploitation virtuelle, aucune des contraintes dures et préférences décrites dans la section 2.2 ne porte sur plusieurs blocs. Ainsi, dans un premier temps nous résolvons chaque bloc indépendamment. Les instances associés au bloc 1 sont B1-LU4, B1-LU8, B1-LU16, B1-LU32 respectivement pour 4, 8, 16 et 32 parcelles élémentaires. Pour toutes les instances du problème, les coûts associés à s-TOP, s-SBC et s-TBC sont respectivement $\delta_1 = 2$, $\delta_2 = 100$ et $\delta_3 = 10$. Ce faisant, nous prenons en compte la hiérarchisation des contraintes décrites dans la section 2.2.2. Afin de privilégier les séquences de culture qui minimisent les effets précédents nous introduisons un facteur $\delta_4 = 10$. Pour tous les effets précédents, k_p (Figure 3), on calcule une nouvelle valeur de $k_p = \delta_4 * k_p$.

Dans un second temps, nous rajoutons une nouvelle préférence qui porte sur l'ensemble de l'exploitation. On définit une nouvelle fonction de coût W_{St}^{SBC} qui étend la précédente (section 2.2). Les proportions annuelles de MA et de BH doivent être respectivement comprises entre $[40, 72]$ ha et $[70, 100]$ ha. Les instances résultantes seront nommées B1[1-4]-LU15(*), B[1-4]-LU30(*), B[1-4]-LU60(*) et B[1-4]-LU120(*). Les blocs sont dans ce cas tous interdépendants. Toutes les instances ci-dessus décrites sont disponibles dans le benchmark *cost function* [1]. Pour chacune des instances, le nombre de contraintes est approximativement égal à $\frac{5}{2} \times \mathcal{N} \times \mathcal{H} \pm 30$, où $\mathcal{N}$ est le nombre de parcelles élémentaires, et $\mathcal{H}$ l'horizon de planification.

5.2 Analyse des résultats

Pour la résolution des PAC, nous utilisons l'algorithme *Depth-First Branch and Bound* (DFBB) implémenté dans le solver **Toulbar2** [2] (version 0.9.1). Le paramétrage utilisé est celui par défaut. Les colonnes $|\mathcal{X}|$ et $|\mathcal{W}|$ du tableau 1 montrent le nombre de variables ($\mathcal{N} \times \mathcal{H}$) et de contraintes pour chaque instance.

Les résultats présentés dans le tableau 1 sont obtenus avec un processeur Intel(R) Xeon(R) de 2.27GHz. Le temps de calcul mentionné est en seconde. Il indique la durée totale pour trouver et prouver l'optimalité (colonne « Temps » de « Une sol. optimale ») en commençant avec un majorant initial relativement bon (colonne UB). Ce majorant influence considérablement la performance. Dans notre cas le choix du majorant initial est empirique. Sur la base de l'optimum trouvé pour la recherche d'une solution, nous recherchons toutes les solutions (colonne « Toutes les sol. optimales ») avec un majorant initial fixé à l'optimum (colonne Opt.) plus un. Pour les instances avec les blocs indépendants, les meilleures solutions sont obtenues en moins d'une minute à l'exception de B1-LU32. Dans ce cas, les solutions optimales sont obtenues et prouvées pour toutes les instances. La différence entre le temps de calcul pour la recherche d'une ou toutes les solutions est principalement liée au majorant initial. Les résultats obtenus en introduisant les interdépendances entre les blocs sont dans l'ensemble acceptables comparativement à la taille des problèmes. En effet, l'arité maximum de certaines contraintes gcc et soft-gcc est égale au nombre total de parcelles élémentaires soit 120 dans le pire des cas. Cela peut expliquer la raison pour laquelle l'instance B[1-4]-LU120(*) n'a pas été résolue au bout de 48 heures.

5.3 Analyse des solutions trouvées pour l'instance B1[1-4]-LU15(*)

De manière générale, les solutions obtenues sont en pratique de très bonnes qualités. Considérons l'ensemble des solutions optimales trouvées pour l'instance B1[1-4]-LU15(*). La Figure 7 montre que pour toutes les années et pour chacune des cultures, les solutions ont le même nombre de parcelles affectées à une culture

[1] http ://www.costfunction.org/benchmark ?task = browse Anonymous&idb=33

[2] http ://mulcyber.toulouse.inra.fr/projects/toulbar2

Tab. 1 – Performance pour la recherche d'une solution optimale et toutes les solutions optimales avec DFBB.

Instance de PAC	$\|\mathcal{X}\|$	UB	$\|\mathcal{W}\|$	Opt.	Une sol. optimale		Toutes les sol. optimales		
					Time(s)	Nodes	Time(s)	Nodes	Nb.Sol
B1-LU4	36	1000	91	92	0.39	17	0.08	8	5
B1-LU8	72	2000	175	184	2.96	94	0.21	32	17
B1-LU16	144	4000	343	368	21.47	413	2.64	256	257
B1-LU32	288	6000	679	640	228	285	6.19	38	17
B2-LU2	18	1000	47	38	0.08	2	0.06	2	1
B2-LU4	36	2000	95	116	0.22	8	0.22	8	1
B2-LU8	72	4000	191	392	4.19	6	0.36	2	1
B2-LU16	144	6000	383	752	7.9	10	0.78	2	1
B3-LU4	36	1000	99	328	0.3	14	0.29	16	2
B3-LU8	72	2000	199	656	0.64	14	0.6	16	2
B3-LU16	144	4000	367	1312	1.51	18	1.37	16	2
B3-LU32	288	6000	703	2592	4.1	20	3.79	18	2
B4-LU5	45	1000	119	46	0.53	4	0.08	0	1
B4-LU10	90	2000	239	192	11.64	5	0.57	0	1
B4-LU20	180	4000	479	752	12.32	12	0.73	0	1
B4-LU40	360	6000	959	1504	39.33	23	1.97	2	1
B[1-4]-LU15(*)	135	2000	360	704	21.02	257	7.87	96	2
B[1-4]-LU30(*)	270	4000	712	1560	323.02	1029	155.9	498	12
B[1-4]-LU60(*)	540	4000	1384	3852	2412.97	1297	3697.23	2228	136
B[1-4]-LU120(*)	1080	8000	2728	-	-	-	-	-	-

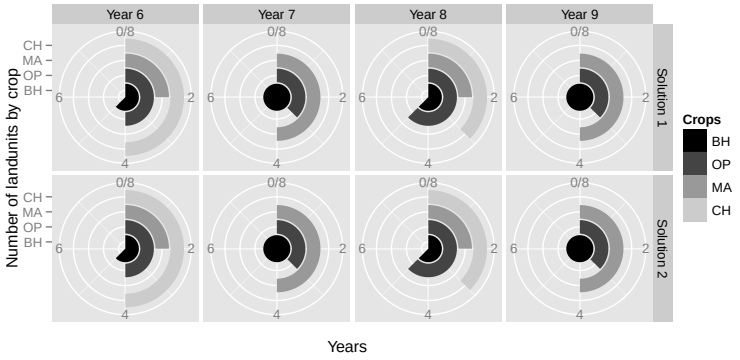


Fig. 7 – Proportion annuelle des cultures pour les différente années.

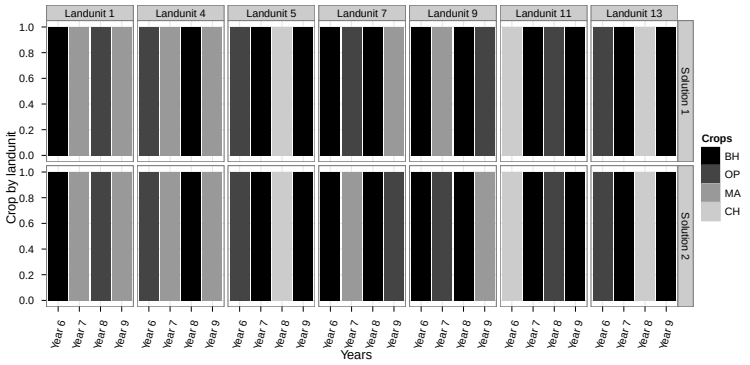


Fig. 8 – Allocation des cultures.

donnée. L'équilibre spatiale du maïs n'est pas respecté pour les années $t \in \{6, 8\}$. Cela est dû aux valeurs d'historique sur le bloc 3.

Par ailleurs, la Figure 8 montre les allocations spatiale et temporelle sur les blocs. Le graphique montre pour chaque solution les séquences de culture sur certaines parcelles élémentaires représentatives : 1, 4 (bloc 1), 5 (bloc 2), 7, 9 (bloc 3) et 11, 13 (bloc 4). On observe que les délais de retour sont respectés. La différence entre les solutions dépend des allocations du bloc 3 (parcelles élémentaires 7 et 9). Pour les années $t \in \{7, 9\}$, l'orge de printemps (OP) peut être substitué par du maïs (MA). Il n'existe aucune symétrie entre les solutions en raison de l'historique et du type de sol.

6 Conclusion

Dans cet article nous avons proposé une modélisation du problème d'allocation de culture (PAC) basée sur les CSP pondérés. Contrairement aux approches existantes, notre proposition est spatialement explicite et intègre les deux dimensions (spatiale et temporelle) du PAC. Nous avons décrit la manière dont les contraintes dures et les préférences d'un agriculteur peuvent être abordées sous la forme d'une optimisation de fonction objective globale. Les résultats obtenus montrent qu'avec le solver *Toulbar2*, des solutions peuvent être trouvées en temps raisonnable pour des PAC de petites et moyennes tailles. Par la suite, nous étudierons (i) l'intérêt des contraintes globales CUMULATIVE pour une prise en compte plus complexe des ressources puis (ii) la fusion des délais de retour et des effets précédents via des contraintes globales de types COSTREGULAR.

Références

[1] J.E. Annetts and E. Audsley. Multiple objective linear programming for environmental farm planning. *Journal of the Operational Research Society*, 53(9) :933–943, 2002.

[2] J Bachinger and P Zander. ROTOR, a tool for generating and evaluating crop rotations for organic farming systems. *European Journal of Agronomy*, 26 :130–143, 2007.

[3] Nicolas Beldiceanu, Irit Katriel, and Sven Thiel. Filtering algorithms for the same constraint. In *CPAIOR*, pages 65–79, 2004.

[4] S. Dogliotti, W. A. H. Rossing, and M. K. van Ittersum. ROTAT, a tool for systematically generating crop rotations. *European Journal of Agronomy*, pages 239–250, 2003.

[5] Jérôme Dury, Noémie Schaller, Frédérick Garcia, Arnaud Reynaud, and Jacques Eric Bergez. Models to support cropping plan and crop rotation decisions. a review. *Agronomy for Sustainable Development*, July 2011.

[6] Talaat El-Nazer and Bruce A. McCarl. The Choice of Crop Rotation : A Modelling Approach and Case Study. *American Journal of Agricultural Economics*, 68(1) :127–136, 1986.

[7] E. O Heady. The Economics of Rotations with Farm and Production Policy Applications. *Journal of Farm Economics*, pages 645–664, 1948.

[8] Willem Jan van Hoeve, Gilles Pesant, and Louis-Martin Rousseau. On global warming : Flow-based soft global constraints. *J. Heuristics*, pages 347–373, 2006.

[9] S. Irnich and G. Desaulniers. *Shortest Path Problems with Resource Constraints*, chapter 2, pages 33–65. Springer, 2005.

[10] Takeshi Itoh, Hiroaki Ishii, and Teruaki Nanseki. A model of crop planning under uncertainty in agricultural management. *International Journal of Production Economics*, pages 555–558, 2003.

[11] W. K. Kein Haneveld and A. W. Stegeman. Crop succession requirements in agricultural production planning. *European Journal of Operational Research*, 166 :406–429, 2005.

[12] B. Leteinturier, J.L. Herman, F. De Longueville, L. Quintin, and R. Oger. Adaptation of a crop sequence indicator based on a land parcel management system. *Agriculture, Ecosystems & Environment*, 112(4) :324–334, 2006.

[13] P. Meseguer, F. Rossi, and T. Schiex. Soft Constraints Processing. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 9. Elsevier, 2006.

[14] Thomas Nesme, Francoise Lescourret, Stéphane Bellon, and Robert Habib. Is the plot concept an obstacle in agricultural sciences ? a review focussing on fruit production. *Agriculture, Ecosystems and Environment*, pages 133 – 138, 2010.

[15] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In *CP*, pages 482–495, 2004.

[16] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the thirteenth national conference on Artificial intelligence*, pages 209–215. AAAI Press, 1996.

[17] Ruhul Sarker and Tapabrata Ray. An improved evolutionary algorithm for solving multi-objective crop planning models. *Computers and Electronics in Agriculture*, 68(2) :191–199, 2009.

Filtrage de fonctions de coût globales décomposables

D. Allouche[1] C. Bessiere[2] P. Boizumault[3] S. de Givry[1]
P. Gutierrez[4] S. Loudni[3] JP. Métivier[3] T. Schiex[1]
[1]UBIA UR 875, INRA, F-31320 Castanet Tolosan, France
[2]U. Montpellier, France
[3]GREYC-CNRS, UMR 6072, U. Caen, France
[4]IIIA-CSIC, U. Autònoma de Barcelona, Bellaterra, Spain

Abstract

As [18] have shown, weighted constraint satisfaction problems can benefit from the introduction of global cost functions, leading to a new Cost Function Programming paradigm. In this paper, we explore the possibility of decomposing global cost functions in such a way that enforcing soft local consistencies on the decomposition achieves the same level of consistency on the original global cost function. We give conditions under which directional arc consistency and virtual arc consistency offer such guarantees. We conclude by experiments on decomposable cost functions showing that decompositions may be very useful to easily integrate efficient global cost functions in solvers.

Résumé

Les auteurs de [18] ont montré que les problèmes de satisfaction de contraintes pondérées peuvent bénéficier de l'introduction des fonctions de coût globales, conduisant au nouveau paradigme de la programmation par fonctions de coûts. Dans cet article, nous explorons la possibilité de décomposer les fonctions de coût globales de sorte qu'appliquer une consistance locale souple sur la décomposition produit le même niveau de consistance que sur la fonction de coût initiale. Nous donnons des conditions pour lesquelles l'arc consistance directionelle et l'arc consistance virtuelle offrent de telles garanties. Nos expérimentations, menées sur des fonctions de coût décomposables, montrent que les décompositions peuvent être très utiles pour intégrer efficacement des fonctions de coût globales dans des solvers.

†. Ce travail a été soutenu par l'Agence nationale de la Recherche, reference ANR-10-BLA-0214.

1 Introduction

Le traitement de modèles graphiques est un problème central en intelligence artificielle. L'optimisation d'une combinaison de fonctions de coût locales, très étudiée dans les réseaux de contraintes pondérées [25], permet de capturer une grande variété de problèmes, tels que SAT, Max-SAT, CSP pondérés, recherche d'une explication de probabilité maximale (MPE pour *Maximum Probability Explanation*) dans les réseaux Bayésiens, le calcul d'un maximum *a posteriori* (MAP pour *Maximum A posteriori Problem*) dans les champs markoviens [14] ou encore l'optimisation de fonctions pseudo-booléennes [6]. Ses applications sont nombreuses en allocation des ressources et en bioinformatique.

La principale approche utilisée pour résoudre ces problèmes s'appuie sur les méthodes fondées sur le principe de séparation et évaluation, combiné avec des techniques dédiées de calcul de minorants permettant d'élaguer l'arbre de recherche. Ces minorants peuvent être fournis par l'application de cohérences locales souples [7], comme dans les solveurs de Programmation par Contraintes (CP). De tels solveurs disposent de contraintes globales qui représentent un outil indispensable pour la modélisation et la résolution de problèmes difficiles de grande taille. Des algorithmes dédiés pour le filtrage de ces contraintes ont été proposés. Pour certaines contraintes globales telles que, Regular, Contiguity, Among, il a été démontré qu'une décomposition en un réseau Berge-acyclique de con-

traintes d'arité fixe peut conduire à une mise en œuvre simple, sans aucune perte d'efficacité du filtrage [2, 4].

La notion de contrainte globale a été récemment étendue aux CSP pondérés, définissant les fonctions de coût globales [29, 28, 20, 19] avec des algorithmes de filtrage efficaces. Dans cet article, après un rappel de quelques notions préliminaires, nous introduisons la notion de décomposition de fonctions de coût globales et nous montrons comment les contraintes globales décomposables peuvent être relâchées sous la forme de fonctions de coût globales décomposables, ayant la même structure de décomposition. Pour les fonctions de coût globales décomposables Berge-acycliques, nous montrons que l'application de la cohérence d'arc directionnelle ou de la cohérence d'arc virtuelle sur la décomposition produit le même niveau de consistance que sur la fonction de coût de départ. Enfin, nous comparons expérimentalement l'efficacité des versions décomposées et monolithiques de plusieurs fonctions de coût globales et les résultats observés montrent des accélérations importantes en faveur de la décomposition.

2 Préliminaires

2.1 Réseau de fonctions de coût

Un réseau de fonctions de coût (CFN) est une paire (X, W) où $X=\{1, \ldots, n\}$ est un ensemble de n variables et W est un ensemble de fonctions de coût. Chaque variable $i \in X$ a un domaine fini $D_i \in D$ de valeurs qui peuvent lui être affectées. La taille du plus grand domaine est notée d. Une affectation de i à la valeur $a \in D_i$ est notée (i, a). Pour un sous-ensemble de variables $S \subseteq X$, on note D^S le produit cartésien des domaines des variables de S. Pour un n-uplet donné t, $t[S]$ représente la projection habituelle du n-uplet t sur l'ensemble de variables S.

Une fonction de coût $w_S \in W$, de portée $S \subseteq X$, est une fonction $w_S : D^S \mapsto [0, k]$ où k est un coût entier maximum (fini ou non) utilisé pour représenter les affectations interdites (exprimant des contraintes dures). Pour capturer fidèlement les contraintes dures, les coûts sont combinés par l'addition bornée $\oplus$, définie par $\alpha \oplus \beta = \max(k, \alpha + \beta)$. Dans cet article, une *contrainte* dure est donc représentée par une fonction de coût prenant des valeurs dans l'ensemble $\{0, k\}$ seulement. Si $\forall t \in D^S, z_S(t) \leq w_S(t)$, la fonction de coût z_S est appelée une relaxation de w_S, notée $z_S \leq w_S$. Un coût β peut être soustrait d'un coût α plus grand en utilisant l'opération $\ominus$ où $\alpha \ominus \beta$ est égale à $(\alpha - \beta)$ si $\alpha \neq k$. Sinon il est égal à k. Nous supposons qu'il existe, pour chaque variable i, une fonction de coût unaire w_i ainsi qu'une fonction de coût d'arité nulle notée $w_\varnothing$.

Le problème consiste à trouver une affectation complète t de l'ensemble des variables minimisant la combinaison des fonctions de coût $\bigoplus_{w_S \in W} w_S(t[S])$. Le problème de décision, associé à ce problème d'optimisation, est NP-complet. Sa restriction aux variables booléennes et aux contraintes binaires est connue pour être APX-difficiles [21].

Un réseau de contraintes (CN) est un CFN où toutes les fonctions de coût représentent des contraintes dures. Ces fonctions de coût seront appelées contraintes.

2.2 Consistance locale

Les algorithmes de recherche de solutions dans les réseaux de contraintes (CN) utilisent les techniques de renforcement de consistance locale afin de réduire l'espace de recherche. Dans ces réseaux, la cohérence d'arc généralisée (GAC) est le niveau de consistance locale le plus utilisé. Une contrainte c_S est dite GAC, ssi, chaque valeur du domaine de chaque variable de S possède au moins un support dans c_S. Un support dans c_S est un n-uplet $t \in D^S$ tel que $c_S(t) = 0$.

Établir la cohérence d'arc généralisée sur c_S sera désigné par le terme *filtrer* c_S. Habituellement, la résolution de problèmes de minimisation dans les réseaux de fonctions de coût repose sur des méthodes exactes fondées sur le principe de séparation et évaluation combiné avec le calcul de minorants obtenus par renforcement de consistances locales telles que la cohérence d'arc directionnelle (DAC) [8, 17] et la cohérence d'arc virtuelle (VAC) [7].

2.3 Fonction de coût globale

Une *contrainte* globale $c(S, \theta)$ représente une famille de contraintes ayant une sémantique donnée, définie sur un ensemble de variables S et incluant d'éventuels paramètres additionnels représentés par θ. Les contraintes globales utilisent des algorithmes de filtrage dédiés plus efficaces que les algorithmes de filtrage génériques. Plusieurs contraintes globales ont été étendues pour capturer une mesure de violation comme par exemple SoftAllDiff(S) [23] ou SoftRegular($S, \mathcal{A}, d$) [27]). Ces contraintes globales relâchable sont des *contraintes dures*, incluant une variable de coût permettant de quantifier la violation selon une sémantique de violation donnée. Pour plusieurs de ces contraintes, des algorithmes efficaces dédiés établissant la GAC ont été proposés.

Récemment, différents travaux [28, 18] ont montré qu'il est possible de capturer ces même mesures de violation au travers de fonctions de coût paramétrées $z(S, \theta)$, calculant directement le coût d'une affectation.

Cette approche permet d'exploiter, de manière immédiate, les consistances locales souples grâce à des algorithmes de filtrage dédiés fournissant des minorants de bien meilleurs qualité.

En effet, les fonctions de coût, combinées avec les consistances locales souples, offrent un filtrage plus performant que l'approche précédente basée sur les variables de coût et les contraintes globales souples. Ces améliorations sont dues à une meilleure communication entre les fonctions de coût permise par l'application de transformations préservant l'équivalence (EPT) [9, 18].

2.4 Hypergraphe

L'hyper-graphe d'un réseau de fonctions de coût (ou d'un réseau de contraintes) (X, W) est constitué d'un sommet par variable $i \in X$ et d'une hyper-arête par portée S telle que $\exists w_S \in W$. Nous ne considérons que des CFN constitués d'hyper-graphes connectés. Le graphe d'incidence d'un hyper-graphe (X, E) est un graphe $G = (X \cup E, E_H)$ où $(x_i, e_j) \in E_H$ ssi $x_i \in X, e_j \in E$ et x_i appartient à hyper-arête e_j. Un hyper-graphe (X, E) est Berge acyclique ssi son graphe d'incidence est acyclique.

3 Décomposition de fonctions de coût globales

Certaines contraintes globales peuvent être efficacement décomposées en un sous-réseau équivalent de contraintes d'arité bornée [5, 3]. De même, les fonctions de coût globales peuvent être décomposées en un ensemble de fonctions coûts d'arité bornée. À noter que la définition ci-dessous s'applique à toute fonction de coût, y compris aux contraintes (fonctions de coût utilisant uniquement les coûts $\{0, k\}$).

Définition 1 *Une décomposition d'une fonction de coût $z(T, \theta)$ est une transformation polynomiale δ_p (p étant un entier) qui retourne un CFN $\delta_p(T, \theta) = (T \cup E, F)$ tel que $\forall w_S \in F, |S| \leq p$ et $\forall t \in D^T, z(T, \theta)(t) = \min_{t' \in D^{T \cup E}, t'[T]=t} \bigoplus_{w_S \in F} w_S(t'[S])$.*

Nous supposons, sans aucune perte de généralité, que chaque variable supplémentaire $i \in E$ figure dans au moins deux fonctions de coût dans la décomposition. [1] Clairement, si $z(T, \theta)$ apparaît dans un CFN $P = (X, W)$ et se décompose en $(T \cup E, F)$, alors les solutions optimales de P peuvent être directement obtenues par projection des solutions optimales du CFN $P' = (X \cup E, W \setminus \{z(T, \theta)\} \cup F)$ sur X.

Exemple Considérons la contrainte AllDiff(S) ainsi que sa version souple SoftAllDiff(S, dec) avec comme mesure de violation la sémantique de *décomposition* [23] où le coût d'une affectation est le nombre de couples de variables ayant la même valeur. Il est bien connu que AllDiff se décompose en un ensemble de $\frac{n.(n-1)}{2}$ contraintes binaires de différence. De manière analogue, la fonction de coût globale Soft-AllDiff(S, dec) peut être décomposée en un ensemble de $\frac{n.(n-1)}{2}$ fonctions de coût de différence. Une fonction de coût de différence prend la valeur 1, ssi, les deux variables impliquées ont la même valeur et 0 sinon. Cette décomposition ne nécessite aucune variable supplémentaire. Enfin, notons que les deux décompositions ont la même structure d'hyper-graphe.

3.1 Relaxation de contraintes globales décomposables

Nous allons à présent montrer qu'il existe une façon systématique d'obtenir des fonctions de coût décomposables, vues comme des relaxations spécifiques de contraintes globales décomposables existantes.

Comme l'a montré le précédent exemple avec AllDiff, si l'on considère une contrainte globale décomposable, alors il est possible de définir une fonction de coût globale décomposable, en relaxant chacune des contraintes issues de la décomposition.

Théoreme 1 *Soit $c(T, \theta)$ une contrainte globale qui se décompose en un réseau de contraintes $(T \cup E, C)$ et f_θ une fonction qui fait correspondre à chaque $c_S \in C$ une fonction de coût w_S telle que $w_S \leq c_S$. Alors, la fonction de coût globale $w(T, f_\theta)(t) = \min_{t' \in D^{T \cup E}, t'[T]=t} \bigoplus_{c_S \in C} f_\theta(c_S)(t'[S])$ est une relaxation de $c(T, \theta)$.*

Preuve Pour chaque n-uplet $t \in D^T$, si $c(T, \theta)(t)=0$, alors $\min_{t' \in D^{T \cup E}, t'[T]=t} \bigoplus_{c_S \in C} c_S(t'[S]) = 0$ car $(T \cup E, C)$ est une décomposition de $c(T, \theta)$. Soit $t' \in D^{T \cup E}$ le n-uplet pour lequel le minimum est atteint. Ceci entraîne que $\forall c_S \in C, c_S(t'[S]) = 0$. Comme $f_\theta(c_S)$ est une relaxation de c_S, ceci entraîne que $f_\theta(c_S)(t'[S])=0$. Ainsi, $\bigoplus_{c_S \in C} f_\theta(c_S)(t'[S]) = 0$ et $w(T, f_\theta)(t) = 0$. □

Par définition, la fonction de coût globale $w(T, f_\theta)$ est décomposable en $(T \cup E, W)$ où W est obtenu par l'application de de f_θ sur chaque élément de C. Il est à noter que, puisque f_θ préserve les portées, l'hypergraphe de la décomposition est lui aussi préservé. Ce résultat permet d'obtenir immédiatement un grand nombre de décompositions pour les fonctions de coût

1. Sinon, une telle variable peut être retirée par élimination de variable : pour cela, supprimer i de E et remplacer la fonction w_S, contenant i, par une fonction de coût $\min_i w_S$ sur $S \setminus \{i\}$. Ce qui préserve la Berge-acyclicité.

globales, ceci à partir des décompositions déjà réalisées de contraintes globales telles que AllDiff, Regular, Grammar, Among, Stretch. La paramétrisation grâce à f_θ permet une grande flexibilité.

Considérons la contrainte AllDiff(S) décomposée en une clique de contraintes binaires de différence. A partir de n'importe quel graphe $G = (V, E)$, il est possible de définir une fonction de relaxation f_G qui préserve les contraintes de différence $i \neq j$ lorsque $(i, j) \in E$, et qui, sinon les relaxe en une fonction de coût constante et égale à zéro. La fonction de coût globale ainsi construite $w(V, f_G)$ permet de modéliser le problème de coloration des sommets d'un graphe qui est un problème NP-difficile. Ainsi, établir DAC ou VAC sur cette seule fonction de coût sera aussi un problème *intraitable*, alors qu'établir DAC ou VAC sur sa décomposition en fonctions de coût binaires sera évidemment un problème polynomial mais qui affaiblira le niveau de filtrage établi.

Considérons la contrainte globale Regular($\{X_1, \ldots, X_n\}, \mathcal{A}$), définie par un automate fini $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ où Q est un ensemble d'états, Σ un alphabet, $\delta :: \Sigma \times Q \rightarrow 2^Q$ une fonction de transition, q_0 l'état initial et F l'ensemble des états terminaux. Les auteurs de [4] ont montré que cette contrainte pouvait se décomposer en un réseau de contraintes $(\{X_1, \ldots, X_n\} \cup \{Q_0, \ldots, Q_n\}, C)$ où les variables additionnelles Q_i ont pour domaine Q. L'ensemble de contraintes C contient deux contraintes unaires restreignant Q_0 à $\{q_0\}$ et Q_n à F ainsi qu'une séquence de contraintes ternaires de la forme $c_{\{Q_i, X_{i+1}, Q_{i+1}\}}$ où un triplet (q, s, q') est autorisé, ssi, $q' \in \delta(q, s)$. Une relaxation de cette décomposition consiste à relâcher chacune de ces contraintes. Les contraintes unaires portant sur Q_0 et Q_n seront remplacées par les fonctions de coût unaires λ_{Q_0} et ρ_{Q_n} donnant le coût d'utilisation de chaque état comme état initial ou comme état final. Les contraintes ternaires seront relaxées sous forme de fonctions de coût ternaires $\sigma_{\{Q_i, X_{i+1}, Q_{i+1}\}}$ donnant le coût d'utilisation de n'importe quelle transition (q, s, q'). Cette relaxation correspond précisément à l'utilisation d'un automate fini pondéré $\mathcal{A} = (Q, \Sigma, \lambda, \sigma, \rho)$ [11]. Le coût d'une affectation (mot) est égal, par définition, au coût optimal de la reconnaissance de ce mot par l'automate pondéré. On obtient ainsi la fonction de coût globale WeightedRegular($\{X_1, \ldots, X_n\}, \mathcal{A}$). Comme l'ont montré les auteurs de [13], la distance de Hamming et la distance d'édition (Edit) peuvent être toutes deux capturées par un automate pondéré. Nous verrons que, contrairement à celle de AllDiff, la décomposition de WeightedRegular peut être traitée de manière efficace par les consistances locales souples.

4 Cohérence locale et décompositions

L'utilisation de décompositions au lieu de leur version monolithique offre des avantages et des inconvénients. Du fait de la localité des calculs, la version décomposée peut être filtrée plus efficacement mais cela peut aussi limiter l'effet du filtrage. Dans les CSP classiques, il est bien connu que si la décomposition a une structure Berge-acyclique, l'établissement de la cohérence d'arc généralisée (GAC) sur la décomposition établit également GAC sur la contrainte globale [1]. Nous montrons qu'un résultat similaire peut être obtenu pour les réseaux de fonctions de coût en utilisant DAC ou VAC.

DAC a été originellement introduit sur les fonctions de coût binaires en s'appuyant sur la notion de support complet [7]. Pour une fonction de coût w_S, un n-uplet $t \in D^S$ est un support complet pour une valeur (i, a) de $i \in S$ ssi $w_i(a) = w_S(t) \bigoplus_{j \in S} w_j(t[j])$. Remarquez que soit $w_i(a) = k$ et (i, a) ne participe à aucune solution ou $w_i(a) < k$ et alors $w_S(t) \bigoplus_{j \in S, j \neq i} w_j(t[j]) = 0$. DAC a été étendu aux fonctions de coût non binaires dans [24] et [20] avec des définitions différentes qui coïncident néanmoins dans le cas de fonctions de coût binaires. Dans cet article, nous nous appuyons sur une extension simple appelée T-DAC (pour Terminal DAC). Étant donné un ordre total $\prec$ sur les variables, un CFN est dit T-DAC par rapport à l'ordre $\prec$ ssi, pour toute fonction de coût w_S, toute valeur (i, a) de la variable maximum $i \in S$ selon $\prec$ a un support complet sur w_S.

VAC est une cohérence locale plus récente qui établit un lien entre un CFN $P = (X, W)$ et un réseau de contrainte dénoté $Bool(P)$ qui a le même ensemble X de variables et qui contient exactement, pour chaque fonction de coût $w_S \in W, |S| > 0$, une contrainte c_S avec la même portée qui interdit précisément tous les n-uplets $t \in D^S$ tels que $w_S(t) \neq 0$. Un CFN P est dit VAC ssi la fermeture arc cohérente du réseau de contraintes $Bool(P)$ n'est pas vide [7].

4.1 Filtrage par cohérences locales souples

L'établissement de cohérences locales souples s'appuie sur des transformations préservant l'équivalence (*Equivalence Preserving Transformations* ou EPT) qui s'appliquent à une fonction de coût [9] w_S. Au lieu d'effacer des valeurs, une EPT déplace des coûts entre w_S et la fonction de coût unaire $w_i, i \in S$. Elle opère donc sur un sous-réseau de P défini par w_S dénoté $N_P(w_S) = (S, \{w_S\} \cup \{w_i\}_{i \in S})$. La principale EPT est décrite comme l'Algorithme 1. Cette EPT déplace une quantité de coût $|\alpha|$ entre la fonction de coût unaire w_i et la fonction de coût w_S. La direction de déplacement des coûts est indiquée par le signe de α. Les précondi-

tions garantissent que les coûts restent positifs dans le réseau obtenu après application.

Algorithme 1 : Une opération de transformation (EPT) permettant de déplacer des coûts pour établir les cohérences d'arc souples. Les opérations $\oplus, \ominus$ sont étendues pour accepter des coûts négatifs comme suit : pour des coûts non négatifs α, β, on a $\alpha \ominus (-\beta) = \alpha \oplus \beta$ et pour $\beta \leq \alpha$, $\alpha \oplus (-\beta) = \alpha \ominus \beta$.

```
Précondition : −w_i(a) ≤ α ≤ min_{t∈D^S, t[i]=a} w_S(t);
Procedure Project(w_S, i, a, α)
    w_i(a) ← w_i(a) ⊕ α;
    pour chaque (t ∈ D^S tel que t[i] = a) faire
        w_S(t) ← w_S(t) ⊖ α;
```

Pour établir T-DAC sur une fonction de coût w_S, il suffit de d'abord déplacer les coûts issus de toute fonction unaire $w_i, i \in S$ dans w_S en appliquant $\mathsf{Project}(w_S, i, a, -w_i(a))$ pour toute valeur $a \in D_i$. Soit j la variable maximum dans in S selon $\prec$, il est alors possible d'appliquer $\mathsf{Project}(w_S, j, b, \alpha)$ pour toute valeur (j, b) et $\alpha = \min_{t \in D^S, t[j]=b} w_S(t)$. Soit t un n-uplet où ce minimum est atteint. t est alors un support complet pour (j, b) : $w_j(b) = w_S(t) \bigoplus_{i \in S} w_i(t[i])$. Ce support peut uniquement être brisé si une fonction de coût unaire $w_i, i \in S, i \neq j$, $w_i(a)$ augmente pour une valeur (i, a).

Pour filtrer le CFN complet (X, W) via T-DAC, il suffit de trier W selon $\prec$ et d'appliquer le processus précédent, successivement sur chacune des fonctions de coût. Quand une fonction de coût w_S est traitée, toutes les fonctions de coût dont la variable maximum apparaît avant la variable maximum de S ont déjà été traitées ce qui garantit qu'aucun des support complets déjà établis ne pourra être brisé dans le futur. Le filtrage par T-DAC s'effectue donc en $O(ed^r)$ en temps, où $e = |W|$ et $r = \max_{w_S \in W} |S|$. En utilisant la structure de données Δ introduite dans [7], la complexité spatiale peut être réduite en $O(edr)$.

L'algorithme le plus efficace pour établir VAC [7] établit en fait une approximation de VAC appelée VAC_ε avec une complexité temporelle en $O(\frac{ekd^r}{\varepsilon})$ et une complexité spatiale en $O(edr)$. De façon alternative, la cohérence d'arc souple optimale (OSAC) peut être utilisée pour établir VAC en temps $O(e^{6.5}d^{(3r+3.5)} \log M)$ (où M est le plus grand coût fini dans le réseau).

4.2 Berge acyclicité et cohérence d'arc directionnelle

Dans cette section, nous montrons que le filtrage par T-DAC d'une décomposition Berge-acyclique d'une fonction de coût ou de la fonction de coût globale d'origine fournissent la même distribution de coût sur la dernière variable et donc le même minorant (fourni par l'établissement de la cohérence de nœud [16]).

Théoreme 2 *Si une fonction de coût globale $z(T, \theta)$ se décompose en un CFN de structure Berge-acyclique $N = (T \cup E, F)$ alors il existe un ordre sur $T \cup E$ tel que la fonction de coût unaire w_{i_n} sur la dernière variable i_n produite par filtrage via T-DAC du sous-réseau $(T, \{z(T, \theta)\} \cup \{w_i\}_{i \in T})$ est identique à la fonction de coût unaire w'_{i_n} produite par filtrage via T-DAC de la décomposition $N = (T \cup E, F \cup \{w_i\}_{i \in T})$.*

Preuve Considérons le réseau décomposé N et $I_N = (T \cup E \cup F, E_I)$ son graphe d'incidence. On sait que I_N est un arbre dont les sommets sont les variables et les fonctions de coût de N. Nous enracinons I_N en une variable de T. Les voisins (parents et fils, s'ils existent) d'une fonction de coût w_S sont les variables de S. Les voisins d'une variable i sont les fonctions de coût impliquant i. Considérons un ordre topologique arbitraire des sommets de I_N. Cet ordre induit un ordre sur les variables $(i_1, \ldots, i_n), i_n \in T$ qui est utilisé pour établir T-DAC sur N. Remarquez que pour toute fonction de coût $w_S \in F$, la variable parent de w_S dans I_N apparaît après toutes les autres variables de S.

Considérons une valeur (i_n, a) de la racine. Si $w_{i_n}(a) = k$, alors tout affectation complète qui étend cette valeur a un coût égal à $w_{i_n}(a)$. Sinon, $w_{i_n}(a) < k$. Soit w_S un fils quelconque de i_n et t_S un support complet de (i_n, a) sur w_S. On a $w_{i_n}(a) = w_S(t) \bigoplus_{i \in S} w_i(t[i])$ ce qui prouve que $w_S(t) = 0$ et $\forall i \in S, i \neq i_n, w_i(t[i]) = 0$. I_N étant un arbre, il est possible d'utiliser le même raisonnement récursivement sur tous les descendants de i_n jusqu'aux feuille de l'arbre. On prouve ainsi que l'affectation (i_n, a) peut être étendue en une affectation complète de coût $w_{i_n}(a)$ dans N. Dans chacun des cas, $w_{i_n}(a)$ est le coût d'une extension optimale de (i_n, a) dans N.

Supposons maintenant que l'on établisse T-DAC en utilisant l'ordre des variables précédent sur le réseau non-décomposé $(T, \{z(T, \theta)\} \cup \{w_i\}_{i \in T})$. Soit t un support complet de (i_n, a) sur $z(T, \theta)$. Par définition $w_{i_n}(a) = z(T, \theta) \bigoplus_{i \in T} w_i(t[i])$ ce qui montre que $w_{i_n}(a)$ est égal au coût d'une extension optimale de (i_n, a) sur $(T, \{z(T, \theta)\} \cup \{w_i\}_{i \in T})$. Par définition d'une décomposition, étant donné que $i_n \notin E$, ce coût est égal au coût d'une extension optimale de (i_n, a) dans N. □

T-DAC est donc assez puissante pour traiter des décompositions Berge-acyclique sans perdre en terme de puissance de filtrage , pourvu qu'un ordre idoine soit utilisé pour appliquer les EPT.

4.3 Berge acyclicité et cohérence d'arc virtuelle

La cohérence d'arc virtuelle offre un lien simple et direct entre réseaux de contraintes et réseaux de fonctions de coût qui permet d'étendre des propriétés classiques des CSP aux CFN sous des conditions simples.

Théoreme 3 *Dans un CFN, si une fonction de coût $z(T,\theta)$ se décompose en un CFN Berge-acyclique $N = (T \cup E, F)$ alors le filtrage via VAC du réseau $(T, \{z(T,\theta)\} \cup \{w_i\}_{i\in T})$ ou du réseau $(T \cup E, F \cup \{w_i\}_{i\in T})$ fournit le même minorant $w_\varnothing$.*

Preuve L'établissement de VAC sur le CFN $P = (T \cup E, F \cup \{w_i\}_{i\in T})$ ne modifie pas l'ensemble des portées et fournit un problème équivalent P' tel que $Bool(P')$ est Berge-acyclique, une situation dans laquelle la cohérence d'arc est une procédure de décision. Nous pouvons donc directement utiliser la Proposition 10.5 de [7] qui précise que si un CFN P est VAC et si $Bool(P)$ est dans une classe de CSP résolu par la cohérence d'arc, alors P a une solution optimale de coût $w_\varnothing$.

De façon similaire, le réseau $Q = (T, \{z(T,\theta)\} \cup \{w_i\}_{i\in T})$ contient une seule fonction de coût d'arité strictement supérieure à 1 et $Bool(Q)$ sera résolu par l'établissement de la cohérence d'arc. L'établissement de VAC produira donc un CFN qui a également une solution optimale de coût $w_\varnothing$. Les réseaux P et Q ont, par définition d'une décomposition, le même coût optimal. □

5 Résultats expérimentaux

Dans cette partie nous évaluons l'intérêt pratique des décompositions de fonctions de coût. Comparées aux propagateurs monolithiques des fonctions de coût globales, ces décompositions sont faciles à implanter et procurent un bon filtrage. Mais leur efficacité en temps reste à évaluer.

Tous les problèmes utilisés ici ont été résolus en utilisant le solveur de fonctions de coût `toulbar2 0.9.5`[2] avec les pré-traitements désactivé (option `-o -e: -f: -dec: -h: -c: -d: -q:`), et avec un ordre d'affectation des variables et un ordre DAC compatibles avec la structure Berge-acyclique des décompositions. L'ordonnancement dynamique des valeurs choisit les valeurs supports pour la cohérence d'arc existentielle (EAC) en premier [15]. Nous n'utilisons pas de majorant initial. Dans tous les cas nous utilisons le même niveau de cohérence locale (à savoir (weak) EDGAC*, plus fort que T-DAC et qui produit donc un $w_\varnothing$ optimal pour chaque fonction de coût globale). Toutes les expériences ont été faites en utilisant plusieurs cœurs CPU Intel Xeon 2.66 Ghz avec 64GB de RAM.

2. https ://mulcyber.toulouse.inra.fr/projects/toulbar2.

5.1 WeightedRegular aléatoire

Comme dans [22], nous générons des automates aléatoires avec $|Q|$ états et $|\Sigma|$ symboles. Nous sélectionnons aléatoirement 30% des paires possibles $(s, q_i) \in \Sigma \times Q$ et choisissons aléatoirement un état $q_j \in Q$ pour former une transition $\delta(s, q_i) = q_j$ pour chacune de ces paires. L'ensemble F des états finaux est obtenu en sélectionnant aléatoirement 50% des états de Q. La génération aléatoire suit une distribution uniforme.

A partir de chaque automate, nous construisons deux CFN : un qui utilise une fonction de coût SoftRegular monolithique avec distance de Hamming [20] et un autre qui utilise la décomposition Berge-acyclique d'une fonction de coût WeightedRegular équivalente. Pour rendre le problème moins artificiel nous ajoutons à chacun de ces problèmes le même ensemble de contraintes unaires aléatoires (une par variable d'origine, les coûts unaires étant aléatoirement choisis entre 0 et 9). Nous mesurons deux temps : (1) le temps pour charger et filtrer le problème initial, et (2) le temps total pour résoudre le CFN (incluant le temps (1)). Le temps (1) donne des informations sur la complexité du filtrage alors que le temps (2) donne des informations sur l'incrémentalité des algorithmes de filtrage. Les temps reportés sont des moyennes sur 100 instances. Les instances atteignant la limite de temps de une heure sont comptés pour 3600 secondes.

n	$\|\Sigma\|$	$\|Q\|$	Monolithique		Décomposé	
			filtrage	solve	filtrage	solve
25	5	10	0,12	0,51	0,00	0,00
		80	2,03	9,10	0,08	0,08
25	10	10	0,64	2,56	0,01	0,01
		80	10,64	43,52	0,54	0,56
25	20	10	3,60	13,06	0,03	0,03
		80	45,94	177,5	1,51	1,55
50	5	10	0,45	3,54	0,00	0,00
		80	11,85	101,2	0,17	0,17
50	10	10	3,22	20,97	0,02	0,02
		80	51,07	380,5	1,27	1,31
50	20	10	15,91	100,7	0,06	0,07
		80	186,2	1 339	3,38	3,47

Si on regarde juste le temps de filtrage, il est clair que les décompositions offrent une très forte accélération malgré une implantation bien plus simple que les propagateurs monolithiques. Le temps de résolution montre que les décompositions héritent aussi de l'excellente incrémentalité des algorithmes standards de consistance locale utilisés sur les contraintes d'arité bornée.

5.2 Nonogrammes

Le problème `prob012` de la CSPLib est un puzzle logique NP-complet dans lequel les cases d'une grille doivent être colorées en blanc ou noir. Cette coloration doit respecter la description de la grille, qui spécifie pour chaque ligne et colonne la longueur de chaque segment noir.

Un nonogramme $n \times n$ peut être représenté en utilisant n^2 variables booléennes x_{ij} qui spécifient la couleur de la case en position (i, j). La contrainte sur les longueurs des segments dans chaque ligne et colonne est exprimée par une contrainte globale REGULAR. Nous avons fait deux types d'expériences sur ces nonogrammes pour évaluer le filtrage des fonctions de coût décomposables.

Les Nonogrames assouplis peuvent être construits à partir de nonogrammes classiques en autorisant la violation de la règle des longueurs des segments noirs. Pour cela nous relaxons les contraintes REGULAR sur chaque ligne et chaque colonne en utilisant une mesure de violation de type "distance de Hamming". Le coût associé indique alors combien de cases doivent être modifiées pour satisfaire la description donnée. Ce problème contient $2n$ fonctions de coût globales WEIGHTEDREGULAR avec des portées se chevauchant. Pour pouvoir appliquer le Théoreme 2 sur chacune de ces fonctions de coût globales on doit trouver un ordre des variables qui est un ordre topologique pour chacune de ces fonctions de coût. Il est facile de produire un tel ordre sur ces problèmes. Les variables x_{ij} peuvent, par exemple, être ordonnées en ordre lexicographique, d'en haut à gauche à en bas à droite,c les variables additionnelles étant insérées n'importe où entre leurs variables originales associées. Les portées des fonctions de coût globales sont souvent le reflet de propriétés définies sur le temps (comme dans les problèmes de planning d'infirmières) ou sur l'espace (comme dans les nonogrammes ou les problèmes de traitement de textes). Dans tous ces cas, l'ordre global induit par le temps ou l'espace définit un ordre des variables qui satisfait souvent les conditions du Théoreme 2.

Pour chaque instance de nonogramme aléatoire $n \times n$, on tire de façon uniforme un nombre de segments entre 1 et $\lfloor \frac{n}{3} \rfloor$ pour chaque ligne et colonne. La longueur de chaque segment est tirée de façon uniforme entre 1 et la longueur maximum que permet la place restante et le nombre de segments restants à placer dans cette ligne ou colonne (en considérant une longueur minimale de 1).

Nous avons résolu ces problèmes avec `toulbar2` comme précédemment et avons mesuré le pourcentage de problèmes résolus ainsi que le temps cpu moyen (les problèmes pas encore résolus au bout d'une heure sont comptés pour une heure) sur des échantillons de 100 instances.

Size	Monolithique		Décomposé	
	Résolus	Temps	Résolus	Temps
6×6	100%	1.98	100%	0.00
8×8	96%	358	100%	0.52
10×10	44%	2,941	100%	30.2
12×12	2%	3,556	82%	1,228
14×14	0%	3,600	14%	3,316

Dans ce contexte plus réaliste impliquant plusieurs fonctions de coût globales qui interagissent, les décompositions sont à nouveau, et de loin, l'approche la plus efficace.

Images avec bruit blanc : On génère une grille solution aléatoire avec chaque case colorée en noir avec une probabilité de 0,5. Une instance de nonogramme est créée à partir de la longueur des segments observés dans cette grille aléatoire. Ces instances ont habituellement plusieurs solutions, parmi lesquelles la grille originale. Nous associons à chaque case un coût unaire aléatoire tiré uniformément entre 0 et 99. Ces coûts représentent le prix du coloriage de la case. Une solution de coût minimum est cherchée. Ce problème a été modélisé en `choco` (version. 2.1.3, options par défaut) et `toulbar2` (option `-h:`) en utilisant $2n$ contraintes globales REGULAR. Dans le modèle `choco`, une contrainte SCALAR impliquant toutes les variables est utilisée pour exprimer le critère à optimiser. Dans `toulbar2`, les coûts de coloriage sont exprimés par des fonctions de coût unaires et les contraintes REGULAR sont représentées par des fonctions de coût WEIGHTEDREGULAR avec les poids dans $\{0, k\}$. La version monolithique a été essayée mais donnait des résultats très mauvais.

Nous avons mesuré le pourcentage d'instances résolues et le temps cpu moyen (les problèmes pas encore résolus au bout de demi-heure sont comptés pour une demi-heure) sur des échantillons de 50 instances.

Size	`choco`		`toulbar2`	
	Résolus	Temps	Résolus	Temps
20×20	100%	1.88	100%	0.93
25×25	100%	14.78	100%	3.84
30×30	96%	143.6	96%	99.01
35×35	80%	459.9	94%	218.2
40×40	46%	1,148	66%	760.8
45×45	14%	1,627	32%	1.321

Sur cette classe de problèmes, appliquer la cohérence souple sur les fonctions de coût globales décomposées est préférable au traditionnel filtrage par borne/GAC cohérence d'un pur modèle CP avec variables de coût. Avec les décompositions, l'utilisation directe des fil-

trages par cohérence molle tels que EDAC, qui implique T-DAC, permet une meilleure exploitation des coûts, avec un effort minime d'implantation.

6 Au delà des fonctons de coût décomposables

Il arrive qu'un problème contienne des fonctions de coût globales non décomposables, c'est à dire que toute décomposition en fonctions de coût d'arité bornée est de taille non polynomiale. Cependant, si le réseau de la décomposition est Berge-acyclique, le Théorème 2 s'applique quand même. Avec de tels réseaux de taille exponentielle, le filtrage prendra un temps exponentiel mais produira de bonnes bornes inférieures. La contrainte globale $\sum_{i=1}^{n} a_i x_i = b$ (a et b sont de petits coefficients entiers) peut facilement être décomposée en introduisant $n-3$ variables additionnelles q_i et des contraintes ternaires de somme de la forme $q_{i-1}+a_i x_i = q_i$ avec $i \in [3, n-2]$ et $a_1 x_1 + a_2 x_2 = q_2$, $q_{n-2} + a_{n-1} x_{n-1} + a_n x_n = b$. Les variables additionnelles q_i ont b valeurs, ce qui est exponentiel en la représentation de b. Nous considérons le problème de *Market Split* comme il est défini dans [10, 26]. Le but est de minimiser $\sum_{i=1}^{n} o_i x_i$ tel que $\sum_{i=1}^{n} a_{i,j} x_i = b_j$ pour chaque $j \in [1, m]$ et x_i sont des variables booléennes (o, a et b sont des coefficients entiers positifs). Nous comparons la décomposition Berge-acyclique dans `toulbar2` et une application directe de `cplex` (version 12.2.0.0) sur le programme linéaire en nombres entiers. Nous avons généré des instances aléatoires avec des coefficients entiers choisis aléatoirement dans $[0, 99]$ pour o et a. $b_j = \lfloor \frac{1}{2} \sum_{i=1}^{n} a_{i,j} \rfloor$. Nous avons utilisé 50 instances avec $m = 4, n = 30$, ce qui donne $\max b_j = 918$. Le nombre moyen de noeuds explorés par `cplex` est 50% plus grand qu'avec `toulbar2`. Mais `cplex` était en moyenne 6 fois plus rapide que `toulbar2` sur ces problèmes. Ce problème de sac à dos 0/1 représente probablement un cas très défavorable à `toulbar2`, étant donné que `cplex` contient l'essentiel de ce qui est connu sur la résolution de problèmes de sac à dos en 0/1 (mais seul une partie de ces résultats s'étend à des domaines plus complexes). Il y a deux voies d'amélioration possible pour `toulbar2` sur ces problèmes. La première serait d'utiliser une combinaison des m contraintes de sac à dos en une seule, comme suggéré dans [26]. Une autre consisterait à exploiter directement les propriétés des contraintes linéaires ternaires pour profiter d'une représentation plus compacte du problème et d'un filtrage plus efficace.

7 Travaux connexes

Il faut remarquer que T-DAC est très proche de la notion de "mini-buckets" [12] et de ce fait, le Théorème 2 peut être aisément adapté pour s'appliquer dans ce cadre. Les 'mini-bucket' appliquent une forme limitée d'élimination de variable : quand une variable x est éliminée, les fonctions de coût qui relient x aux variables restantes sont partitionnées dans des ensembles de fonctions de coûts qui contiennent au plus i variables dans leur portée et au plus m fonctions. Si l'on calcule les 'mini-bucket' avec le même ordre de variables que T-DAC, avec $m = 1$ et une valeur de i non bornée, on obtient la même fonction de coût marginale que T-DAC sur la variable racine r, avec la même complexité temporelle. Les 'mini-bucket' peuvent être mis en œuvre de deux façons différentes : la variante statique ne nécessite aucune mise à jour durant la recherche mais restreint l'ordre d'instanciation des variables à un ordre statique ; la variante dynamique permet d'utiliser un ordre d'affectation des variables dynamique (DVO) mais souffre d'un manque d'incrémentalité. Les cohérences locales souples, du fait qu'elles s'appuient sur la notion d'EPT, fournissent toujours un problème équivalent, offrant ainsi une incrémentalité totale durant la recherche ainsi qu'une compatibilité parfaite avec les ordres d'affectation dynamique des variables. Les cohérences d'arc souples offrent également une complexité spatiale en $O(edr)$ alors que les 'mini-bucket' peuvent demander un espace exponentiel en i.

8 Conclusion

Dans ce papier, nous avons étendu la décomposition de contraintes aux fonctions de coût apparaissant dans des CFN. Pour des fonctions de coût ayant une décomposition Berge-acyclique, nous avons montré qu'un filtrage simple tel que la cohérence d'arc directionnelle, fournit un filtrage sur la décomposition comparable à celui de la fonction de coût globale elle-même (à condition de fournir un ordre idoine pour les variables). Les résultats que nous obtenons pour le filtrage fourni par la cohérence d'arc virtuelle (plus fort que DAC) sont identiques et ne nécessitent pas d'ordre particulier.

L'application de ces résultats sur la classe triviale des contraintes globales Berge-acycliques ou *fonctions de coût Berge-acycliques décomposables*, est déjà très significative, car elle permet d'établir des cohérences locales souples sur des réseaux contenant des fonctions de coût Berge-acycliques décomposables telles que Regular, Grammar, Among,...

Nous avons montré que pour ces contraintes globales Berge-acycliques peuvent aussi être relaxées en des

fonctions de coût globales Berge-acycliques généralisant la mesure de violation basée sur la « décomposition ». Ce résultat fournit une longue liste de fonctions de coût Berge-acycliques décomposables. Nos résultats expérimentaux basés sur l'utilisation de T-DAC sur la relaxation de la contrainte REGULAR en la fonction de coût WEIGHTEDREGULAR montre que cette approche basée sur la décomposition permet une importante diminution du temps de calcul par rapport au propagateur monolithique. De plus, l'utilisation de telles fonctions de coût simplifie grandement l'implémentation par rapport aux versions monolithiques de ces fonctions de coût.

Afin d'évaluer expérimentalement l'intérêt pratique de l'utilisation de VAC, il est nécessaire d'implémenter techniquement VAC sur des fonctions de coût non binaires.

Bien que les travaux présentés ici soient restreint aux décompositions Berge-acycliques, ceux-ci ouvrent la voie vers une forme plus générale de « décomposition structurelle » des fonctions de coût globales décomposables en une structure acyclique composée de fonctions de coût locales, possédant des séparateurs de taille bornée (pas nécessairement de taille 1). Ces fonctions de coût globales décomposées pourraient alors être filtrées efficacement en utilisant des transformations préservant l'équivalence (dédiées et incrémentales) basées sur des algorithmes issus de la programmation dynamique.

Références

[1] C. Beeri, R. Fagin, D. Maier, and M.Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30 :479–513, 1983.

[2] N. Beldiceanu, M. Carlsson, R. Debruyne, and T. Petit. Reformulation of global constraints based on constraints checkers. *Constraints*, 10(4) :339–362, 2005.

[3] C. Bessiere. Constraint propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 3. Elsevier, 2006.

[4] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Slide : A useful special case of the cardpath constraint. In *Proc. of ECAI'08*, pages 475–479, 2008.

[5] C. Bessiere and P. Van Hentenryck. To be or not to be ... a global constraint. In *Proc. CP'03*, pages 789–794, 2003.

[6] E. Boros and P. Hammer. Pseudo-Boolean Optimization. *Discrete Appl. Math.*, 123 :155–225, 2002.

[7] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174 :449–478, 2010.

[8] M C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3) :311–342, 2003.

[9] M C. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2) :199–227, 2004.

[10] Gérard Cornuéjols and Milind Dawande. A class of hard small 0-1 programs. In *Integer Programming and Combinatorial Optimization, 6th International IPCO Conference, Houston, Texas, USA, June 22-24, 1998, Proceedings*, pages 284–293, 1998.

[11] Karel Culik II and Jarkko Kari. Image compression using weighted finite automata. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *MFCS*, volume 711 of *Lecture Notes in Computer Science*, pages 392–402. Springer, 1993.

[12] Rina Dechter. Mini-buckets : A general scheme for generating approximations in automated reasoning. In *Proc. of the* 16^{th} *IJCAI*, pages 1297–1303, 1997.

[13] George Katsirelos, Nina Narodytska, and Toby Walsh. The weighted grammar constraint. *Annals OR*, 184(1) :179–207, 2011.

[14] D. Koller and N. Friedman. *Probabilistic graphical models.* MIT press, 2009.

[15] J. Larrosa, S. de Givry, F. Heras, and M. Zytnicki. Existential arc consistency : getting closer to full arc consistency in weighted CSPs. In *Proc. of the* 19^{th} *IJCAI*, pages 84–89, Edinburgh, Scotland, August 2005.

[16] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proc. of the* 18^{th} *IJCAI*, pages 239–244, Acapulco, Mexico, August 2003.

[17] Javier Larrosa and Thomas Schiex. Solving weighted CSP by maintaining arc consistency. *Artif. Intell.*, 159(1-2) :1–26, 2004.

[18] JHM. Lee and KL. Leung. Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction. *Journal of Artificial Intelligence Research*, 43 :257–292, 2012.

[19] Jimmy Lee and K. L. Leung. A stronger consistency for soft global constraints in weighted constraint satisfaction. In Maria Fox and David Poole, editors, *Proc. of AAAI'10.* AAAI Press, 2010.

[20] Jimmy Ho-Man Lee and Ka Lun Leung. Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. In Craig Boutilier, editor, *Proc of the* 21^{th} *IJCAI*, pages 559–565, 2009.

[21] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3) :425–440, 1991.

[22] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2004.

[23] Thierry Petit, Jean-Charles Régin, and Christian Bessiere. Specific filtering algorithms for over-constrained problems. In *CP*, pages 451–463, 2001.

[24] Martí Sánchez, Simon de Givry, and Thomas Schiex. Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1-2) :130–154, 2008.

[25] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems : hard and easy problems. In *Proc. of the* 14^{th} *IJCAI*, pages 631–637, Montréal, Canada, August 1995.

[26] M. A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operations Research*, 118(1-4) :73–84, 2003.

[27] Willem Jan van Hoeve, Gilles Pesant, and Louis-Martin Rousseau. On global warming : Flow-based soft global constraints. *J. Heuristics*, 12(4-5) :347–373, 2006.

[28] M. Zytnicki, C. Gaspin, S. de Givry, and T. Schiex. Bounds arc consistency for weighted CSPs. *Journal of Artificial Intelligence Research*, 35(2) :593–621, 2009.

[29] M. Zytnicki, C. Gaspin, and T. Schiex. A new local consistency for weighted CSP dedicated to long domains. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC)*, pages 394–398, Dijon, France, April 2006.

Optimal Allocation of Renewable Energy Parks: A Two-stage Optimization Model

Mohammad Atef[1] **Carmen Gervet**[1]

[1] German University in Cairo, Tagamoa El-Khamis, New Cairo city 11835, Egypt
{mohammad.atef,carmen.gervet}@guc.edu.eg

Résumé

La recherche appliquée en énergies renouvelables soulève des défis complexes de nature technologique, économique ou politique. Dans cet article, nous adressons le problème technico-économique de sélection optimale d'emplacements de parcs éoliens et solaires en Egypte, de telle sorte que la demande en électricité soit satisfaite à coût minimum. Ultimement, notre objectif est de construire un outil d'aide à la décision qui va pourvoir aux investisseurs privés et gouvernementaux, une aide précieuse pour prendre des décisions à court et long terme relativement à la création de tels parcs. Les approches existantes raisonnent essentiellement à partir de données passées. Dans cet article, nous introduisons une nouvelle approche qui considère à la fois les données passées et futures, et montrons l'impact de la prise en compte des deux types de données dans une modèle d'optimisation en deux temps. Nous montrons que la Programmation Linéaire par Entiers est une méthode efficace pour résoudre le problème avec des données passées ainsi que le modèle en deux temps qui prend en compte les données prédites, rajoutant de nouvelles contraintes au modèle initial. Notre évaluation montre que le modèle en deux temps améliore la solution avec une vision plus globale et un meilleur coût total.

1 Introduction

The need for clean energy is recognized worldwide not only to face global warming and CO_2 emissions, but also to reduce grounds for international conflicts. National and international targets are being set [2, 3]. There are many aspects to the development of renewable energy technologies which can be broadly categorized into engineering & technological advancement aspects, versus techno/economical and commercial ones. The engineering components deal with the construction of renewable plants that are reliable, effective and realistic including essentially hydro, solar (photo-voltaic and concentrated solar power), wind and bio-fuel.

The techno-economical study of renewable energy on the other hand, investigates gradual implantation of Renewable Energy (RE) systems for a given country such that the installation and maintenance costs are minimized and the short/long term returns on investment are maximized. Studies in this field investigate country profiles in terms of energy demand, available resources, anticipated renewable engineering cost reductions [13]. However, more is needed as highlighted in [11], that "there is little economic analysis of renewable energy". The main objectives of studying the economics of RE is to attract investments (national and international) and set realistic targets and strategies that will remain so in the longer term.

Comprehensive surveys are now available discussing the trends and current improvements in the cost, performance, and reliability of renewable energy systems [2]. Clearly electricity from renewable energy remains generally more expensive than from conventional fossil-fuel sources. However, the cost of electricity from RE sources has been falling steadily for the last two decades and various estimates have been derived in terms of "expected cost of electricity production from RE sources" [5]. Today wind energy is the least expensive option but requires more maintenance, and is space consuming compared to photo-voltaic solar panels which however, are currently more expensive. Note that the forecasts in price reduction are promising though, as shown in Figure 1 [2]. This indicates that taking into account forecast measurements is a strong element of effective decision making.

Based on existing forecast studies, and each country renewable resources, which REs or portfolio of RE should a given nation invest in ? How much should be

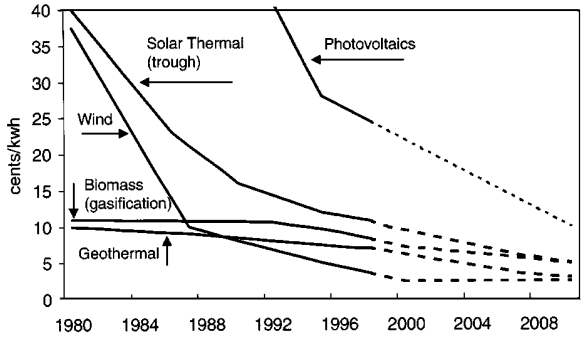


FIGURE 1 – RE costs forecast

invested now, or in 15 yeas time ? These are questions at the heart of the "economics of RE" for which our decision support tool aims to seek an answer. We seek the best trade-off cost/return on investment by taking into account physical installation constraints as well as energy requirements and costs.

In this paper we focus essentially on Egypt, even though the methodology employed can be generalized to other countries.The scene in Egypt today can be summarized briefly as follows, regarding the aspects we are concerned with. Energy consumption in Egypt is growing at fast pace and relies extensively on fossil fuel as shown in the latest earth-trends survey [21]. Egypt enjoys excellent wind and solar resources and there are tremendous potentials for investment towards local consumption and even export. Research and onsite-projects are being carried out with a growing trend.

A short-term governmental plan is to meet 20% of Egypt electricity demand by 2020 using RE sources. While wind farms installation is currently cheaper than solar panels of Concentrated Solar Power (CSP) plants, there are strong arguments in favor of solar energy. Thus it is essential to consider both. Basically solar energy can be installed on small surfaces (little cable and maintenance required), offers more stability especially in Egypt (sun is less dependent on season fluctuations for countries located on the sun belt), plus wasted heat has the advantage of being usable for water desalination [6]. Egypt can become a strong player in wind and solar energy. However a specific strategy is still to be determined, together with investments.

The optimization problem we address is defined as follows. Given the country of Egypt with its data and constraints : 1) Egypt map of populated areas, 2) Wind and Solar atlas, 3) electricity grid map, 4) current and forecast energy cost per RE resource, 5) current and forecast energy demands per month, 6) a set of potential RE park locations ; determine the set of energy parks to be invested in today, the set of energy parks to be invested in the future (e.g. 10-20 years time), such that 20% of the current and forecast energy demand are covered for each month of the year, and the anticipated financial cost is minimized. The cost is determined in terms of sum of total costs associated with each potential park : cost of connection to the grid, cost of installation, and cost of park maintenance.

In this paper, we adopt an iterative development methodology by first focusing on a problem instance with current data only, and extending it to include forecast data and constraints. Thus, both the one-stage and two-stage approaches are presented. We consider three different models and techniques to tackle the one-stage model : dynamic programming, local search and Integer Linear Programming. A comparison is carried out among all the approaches on different benchmarks and randomly generated instances of the problem.We then apply the most efficient technique to solve the two-stage problem and compare the results of both approaches. This approach is a novel contribution by taking into consideration short term and long term impact on the costs. Using a cost forecast to estimate the cost of the different technologies in the future, our model finds which potential parks should be built now to satisfy the current electricity demand, which should be built after ten years to satisfy Egypt's expected electricity demand for the year 2020. To our knowledge this concept has not been considered with respect to renewable energy park placement problems and constitutes the main contribution of this paper. Our contribution lies mainly in investigating the benefits and suitable methods for the two-stage approach.

The paper is organized as follows. In Section 2 we survey related works. In Section 3 we introduce optimization techniques that we use to solve the problem. Sections 4, 5 and 6 describe respectively the different models approaches we investigated. The implementation and experimental results are respectively given in Sections 7 and 8. Finally, a summary of the contribution of this work is given in Section 9 along with suggestions and ideas for future research.

2 Related work

The two-stage approach that takes into account both present and forecast data is novel to our knowledge and has not been addressed with respect to such selection problems. However, variants of the core problem have been tackled with different constraints and objective functions. We summarize them hereafter. The main model is a form of energy resource allocation problem using either a single objective (e.g. minimizing the total annual cost of building renewable energy parks), or multiple objectives of different sorts such as minimizing pollution emissions of CO_2 and

NO_x, maximizing self-production of energy, in addition to the costs factor.

In [17], a survey of different models is presented relative to the problem of energy planning using multi-criteria decision making. While such approaches put a strong emphasis on evaluating the best trade-off between (possibly conflicting) criteria, it does not account for the physical location of parks, nor the future data trends in terms of energy needs and RE sources costs. This could have the downfall that the solution proposed is not technically viable or could be obsolete within few years given the technological advances and cost reductions.

Some approaches address the problem of park placement and selection in specific countries with different objective functions. [15] focuses on the implementation of wind and PV parks to supply electricity in rural areas of Japan. The simulation tool optimizes the cost and seeks to reduce CO_2 emissions. The main drawback is that it only focuses on the present demand and cost values, and ignores the longer term situation. From a different perspective of the problem, [12, 20, 24] focus on the economic dispatch of electricity such that the total fuel cost is minimized, together with the total emissions of CO_2 and NO_x. As such they do not consider the park placement problem but rather the source of RE to consider to reach the objectives. While such problems have raised a lot of interest due to the study of gas emissions, it does not account for the technical aspects that must be taken into account together with the economical ones.

[1] is the most recent and closest work to ours, where the goal of the decision support tool was to increase renewable energy parks, and in turn reduce the usage of non-renewable source. Similar to our problem, it combines the idea of relating the objective of minimizing costs with the choice of physical location of RE parks, but again no account for future reductions in costs and increased demands. It is important though to note that this work showed empirically that a non-fully utilized park is not cost-effective, mainly due to the fact that a great portion of the cost of establishing a RE park is proportional to the distance of the park to the electricity grid, ie. transporting the energy. Thus once the connection is established, one might as well transport as much electricity as possible. We will use this insight in our models.

3 Techniques background

We now briefly recall the foundations of the different techniques we will be evaluating, namely dynamic programming, constrained local search and Integer linear programming.

Dynamic Programming (DP). DP is a technique occasionally used for solving optimization problems [4]. It amounts to making a sequence of decisions that yield an optimal solution given a cost function. However, not all optimization problems are solvable using DP. The problem must have two main properties namely, optimal substructure and overlapping subproblems. If the problem lacks either of these these properties, then it is either unsolvable using DP or inefficient to solve using this technique. The first property ensures that the optimal solution to the problem contains within it an optimal solution to subproblems. By identifying the optimal substructure and thus the subproblems, a recursive formulation of the solution can be defined. The overlapping property ensures that it is worth using DP to solve the problem. When the computation of some states is needed more than once, this is where DP pays off. From an implementation point of view, the time complexity of a DP algorithm can be approximated by the product of the number of subproblems the algorithm goes through overall, and the number of choices necessary to determine which subproblems to use. Examples of problems for which there exists polynomial time DP algorithms are shortest path, sequence alignment, context-free grammar parsing, etc. There are also pseudo-polynomial algorithms to problems such as knapsack, subset sum, cutting-stock [14]. In the next section we show how DP can be used for our problem by reducing it to an instance of the knapsack problem.

Constrained Local search. Similar to many other non-deterministic optimization techniques, Local Search (LS) is a method for searching solution spaces of hard combinatorial problems. The result does not guarantee a globally optimal solution. LS basically makes an educated guess to find an initial solution and then makes a fast enough update to reach a better neighboring solution. The main focus of research in this field lies in finding a "good" neighborhood operator, according to the problem structure and objective function. If the problem is constrained and the solution needs to satisfy such constrained we then talk about constrained local search [22]. This is the case in our design of a local search technique for the optimal park selection problem.

Integer linear programming. Finally we investigated a third approach that requires the problem to be modeled as a linear problem, ie. all the constraints are linear. The Simplex algorithm is one of the most effective methods used to solve LP. The problem at hand is expressed as a maximization (or minimization) of a linear function, such that all the constraints are linear and the variables are positive reals. If any of the variables take integer values, we are then dealing with an

Integer Linear Program (ILP). This will be the case in our study.

4 Optimization Models

We now explore the approaches we considered for the optimal park selection problem. As mentioned earlier, in terms of costs, transportation or connection to the grid is the main bottleneck, thus the problem of selecting a given park, with its maximum capacity, implies that a "yes" is equivalent to "full capacity usage". This reduces the search space subsequently. Thus independently of the techniques we will use, the model will be Boolean in terms of the decision variables. Another observation that applies to our case study of Egypt, is that the electricity grid is fully connected. This implies that the issue is not about selecting certain park locations to cover certain populated areas, it is more global. As such a traditional set covering model is not required and we introduce our *Grid model*.

4.1 The grid model

The following input data is provided by the decision maker to our system, but can also be preprocessed and entered automatically.

Input data

$$C_i = \textit{Cost of park } i \tag{1}$$
$$M_i = \textit{Maximum area of park } i \tag{2}$$
$$(x_i, y_i) = \textit{Coordinates of park } i \tag{3}$$
$$G_{ij} = \textit{Watts/m}^2 \textit{ produced by park } i \textit{ in month } j \tag{4}$$
$$D_j = \textit{20\% Electricity demand in Egypt,month } j\textit{, 2010} \tag{5}$$

Note that the cost C_i is the total cost for each potential location and is composed of, installation cost, maintenance cost, and the cost of building the transportation lines to connect and bring the electricity to the grid.

Decision variables Since a Boolean model is used, the decision variables are associated to each potential park location :

$$\mathbf{B_i} = \begin{cases} 1 & \textit{if park } i \textit{ is chosen} \\ 0 & \textit{otherwise} \end{cases} \tag{6}$$

Constraints The total electricity generated by the RE parks must satisfy 20% of the current demand. Given the demand D_j for month j, we have :

$$\sum_{i=1}^{n} G_{ij} \cdot \mathbf{B_i} \geq D_j \quad \mathbf{B_i} \in \{0,1\} \quad \forall j \in \{1,..,12\} \tag{7}$$

Objective Function We seek to minimize the current cost over all the potential parks :

$$minimize \quad \sum_{i=1}^{n} \mathbf{B_i} \cdot C_i \tag{8}$$

4.2 Multi-dimensional Knapsack equivalence

We now show that the grid model actually formulates the optimal park selection problem as a Multi-Dimensional Knapsack problem (MDK). This implies that we can benefit from techniques suitable to the MDK to solve our problem. The classical Knapsack Problem (KP) is defined by the tuple (n, P, W, Q), where n elements have each a profit P_i and a weight W_i. The objective is to select elements such that the total profit is maximized under the constraint that the total weight should not exceed c. We consider the discrete 0-1 KP. The 0-1 KP maps directly our Boolean grid model. More formally the 0-1 KP is specified by :

$$\text{MAXIMIZE} \sum_{i=1}^{n} P_i \cdot \mathbf{X_i} \tag{9}$$
$$\text{subject to} \sum_{i=1}^{n} W_i \cdot \mathbf{X_i} \leq Q, \quad \mathbf{X_i} \in \{0,1\} \tag{10}$$

where $\mathbf{X_i} = 1$ if and only if the i^{th} element is selected.

The multi-dimensional KP extends it and is specified as a tuple (n, m, P, W, Q) where m is the number of dimensions ; and we have :

$$\text{MAXIMIZE} \sum_{i=1}^{n} P_i \cdot \mathbf{X_i} \tag{11}$$
$$\text{subject to} \sum_{i=1}^{n} W_{ij} \cdot \mathbf{X_i} \leq Q_j, \mathbf{X_i} \in \{0,1\}, \forall j \in \{1,..,m\} \tag{12}$$

where W_{ij} is the weight consumed by the i^{th} element in the j^{th} dimension, and Q_j is the upper limit of the j^{th} dimension. Note that there is one constraint per dimension. A solution to the MKP is one that satisfies the conjunction of the m constraints (12). An optimal solution is one that maximizes (11).

Obviously, the objective of the standard MKP problem is to maximize the profit, whereas in the optimal park selection problem our objective is to minimize the total cost. Also, in the MKP the constraints set an upper bound not to be exceeded for each dimension, whereas in the park selection we have a lower bound to cover the electricity demand for each month. We just need to transform the problem by multiplying constraints and objective by (-1) to have the equivalence. The optimal park selection problem corresponds

to the MKP, whereby $D_j = -Q_j$ is the demand to be covered at month j ; $G_{ij} = -W_{ij}$ is the energy gain of park i during month j ; and $C_i = -P_i$ is the total cost of park i.

5 Algorithms

We now present the different algorithms we implemented to solve this problem. The first is a pseudo-polynomial deterministic algorithm that finds the exact optimal solution using DP. The second is a constrained local search algorithm that uses a neighborhood operator to find a good solution. The third and most successful approach uses Integer Linear Programming.

5.1 Dynamic Programming Algorithm

The most efficient algorithm for solving the KP and MKP is based on DP [14]. We extend it to the the optimal park selection problem. Given n potential park locations to choose from, and assume for simplicity a constant monthly demand of d kWh, we can divide the problem into two independent subproblems. In the first subproblem, the n^{th} location is included in the solution, the cost associated with the n^{th} is subtracted in the new subproblem and the cost is added. In the second subproblem the n^{th} location is not included. The following smaller subproblems solve the same problem with $n-1$ potential locations. More formally, the problem to be solved is defined recursively by :

$$(n, C, G, d) \tag{13}$$

where n is the number of locations, c is the rate of electricity required in kWh, G_n is the production gain of kWh provided by the n^{th} potential location, and C_n is the total cost associated with it. The objective function to minimize is $F(n, d)$ which is defined recursively by :

$$F(n,d) = \begin{cases} 0 \text{ if } d \leq 0 \\ \infty \text{ if } n = 0 \text{ and } d > 0 \\ \min \begin{pmatrix} F(n-1,d) \\ C_n + F(n-1, d-G_n) \end{pmatrix} otherwise \end{cases} \tag{14}$$

Clearly if the demand $d \leq 0$ then the cost is 0, while for strictly positive demand and no potential locations, there is no solution which is represented by an infinite cost value. Otherwise, the solution is the minimum of two subproblems, one that includes the n^{th} location and the other that excludes it.

Complexity This recursive formulation leads to a DP algorithm with a time complexity expressed in terms of : n, the number of potential location, D_j the demand at month j (in $\{1, .., 12\}$), and d, the largest demand to be covered. As aforementioned with respect to knapsack problems, the time complexity is pseudo-polynomial, which is theoretically the best available for the classical KP. The time complexity is in $O(n \cdot d^m)$ where $d = \max_{j=1}^{m} \mathrm{D}_j$.

In our version of the problem m is a constant equal to 12, corresponding to the number of months in this case. The same bound of the time complexity serves as a loose upper bound one the space complexity. This is due to the fact that when top-down DP is used, some subproblems are never reached during solving the original problem. The reason for that is the first base case of recurrence (14), which prunes the computation once the demand has been satisfied. Computational results are presented in the implementation section.

Preprocessing : adding a sorting heuristic A preprocessing step to this approach is to sort the locations before running the DP algorithm. The sorting rule is defined as follows :

$$Location_a < Location_b \iff \sum_{j=1}^{m} G_{aj} < \sum_{i=1}^{m} G_{bj}$$

This causes the locations with higher gain values for electricity rates to come first in the list of locations. Such an ordering forces many branches of the search tree to be pruned early, hence a better performance. Note that this does not affect the optimality of the algorithm. In other words, the sorting heuristic does not affect the completeness of this approach. However, it does make it more efficient in practice. In Section 8, the sorting heuristic is evaluated, where we compare the running times of two versions of the algorithm, one which applies the sorting heuristic, and one which does not.

5.2 Constrained Local Search Algorithm

While the sorting heuristic improves the running time of the DP approach,it is not effective enough to scale up the problem as we illustrate in the experimental results. To improve, we sought a polynomial-time, but sub-optimal algorithm to solve the problem. Now we discuss a local search method used to determine a sub-optimal solution to the problem. The main aspect of local search techniques is to define the neighborhood operator. We use a successful neighborhood operator suggested by Ghosh et al. [9]. Their work focuses on solving the subset sum problem and comparing their suggested neighborhood with previous ones for the same problem. The reason we use a subset sum

problem technique to solve our problem, is that the MKP model is very similar to subset sum with a slight change in the objective function. In MKP the chosen elements in the solution may not exceed the upper bound, but in the subset sum problem the chosen elements must sum up exactly to the bound.

In our work, we modify and implement the idea to solve the optimal park selection problem as a MKP, which requires several changes to [9] that are highlighted here.

Neighborhood operator In order to define the operator we first define the permutation we use and how a solution is specified. We define Y the permutation vector over all potential parks. Instead of reasoning directly about the Boolean vector $\mathbf{X}$, and trying to improve upon a solution using operations on $\mathbf{X}$, we represent a search space of permutations Y of the park location indices. The neighbor of a permutation Y is a vector that differs from Y in only two values (one swap). For example, for $n = 4$, and Y $= [1, 2, 3, 4]$, we have NEIGHBORHOOD(Y) $= \{[\underline{2}, \underline{1}, 3, 4], [\underline{3}, 2, \underline{1}, 4], [\underline{4}, 2, 3, \underline{1}], [1, \underline{3}, \underline{2}, 4], [1, \underline{4}, 3, \underline{2}], [1, 2, \underline{4}, \underline{3}]\}$. However, $[3, 1, 2, 4]$ and $[4, 3, 2, 1] \notin$ NEIGHBORHOOD(Y) since they each contain two swaps.

Algorithm 5.1 GREEDY

Require: G, D, Y
  $\mathbf{E} \leftarrow \{\mathbf{E}_i = 1, \forall 1 \leq 1 \leq n\}$ {solution vector with all parks selected}
  $V \leftarrow [V_1, .., V_m]$such that$V_j = \sum_{i=1}^{n} G_{ij}$
  {vector of aggregated energy covered by all parks per month j}
  for $i = 1 \rightarrow n$ **do**
    if $E - G_{Y_i} \geq D$ **then**
      $E \leftarrow E - G_{Y_i}$
      $\mathbf{V}_{Y_i} \leftarrow 0$
    end if
  end for
  return E

The local search algorithm operates as follows : choose an initial permutation randomly, use the greedy heuristic to compute the solution that matches the permutation with a good cost while satisfying the constraints (GREEDY(G,D,Y)), compute the set of neighboring vectors, determine if a neighbor point can lead to a better solution cost (FINDBEST(Y)). If one exists, move to this permutation with associated solution, if not exit.

The key point lies in the heuristic used to compute a good solution for a given permutation. The greedy algorithm starts by including the entire list E of potential locations in the initial solution. Given a permutation Y of the potential locations, the algorithm traverses the potential locations according to the ordering of Y. For every potential location, if removing it from E causes the total demand to be unsatisfied, the algorithm does not remove it, otherwise it is removed.

Algorithm 5.2 Local Search

Require: G, C, D
  Y $\leftarrow [Y_1, Y_2, ..., Y_n]$ {*random permutation*}
  X $\leftarrow$ GREEDY(G, D, Y)
  loop
    Y' $\leftarrow$ FINDBEST(Y) {*finds the best permutation in the neighborhood of Y which is better than Y*}
    if Y' does not exist **then**
      return X
    end if
    X $\leftarrow$ GREEDY(G, D, Y') {*binary solution from the greedy algorithm (5.1)*}
  end loop

The local search algorithm moves away from an initial permutation Y by searching for the best neighboring permutation Y', that corresponds to the satisfiable neighbor with the smallest cost. The algorithm terminates after failing to find a better neighbor permutation to switch to. Typically, a local search algorithm is run for a number of iterations, and the final result is the best solution over all solutions which the algorithm terminated with during all the iterations. At every new iteration in our implementation, the algorithm is not permitted to re-visit permutations that were already encountered during previous iterations [1].

Complexity The highest cost of Algorithm 5.2 is FINDBEST, which finds the best next solution, better than the current one from within the neighborhood. The time complexity of FINDBEST is polynomial in the number of locations.

$$O(n^3 \cdot m) \tag{15}$$

where n is the number of potential locations, and m is the number of seasons (12). Therefore, m is a constant and the complexity can be reduced to $O(n^3)$. The time complexity of the full algorithm depends on how many iterations the loop in Algorithm (5.2) does. This is controlled by how far is the initial random permutation from a local minimum, since the algorithm terminates when no better solution is found in the neighborhood.

1. This is called local search with memory, which resembles the operation of tabu search.

5.3 Integer Linear Approach

Finally we tried a third approach which proved the most effective in terms of both, solution quality and efficiency. The constraints in Equation (7) are linear constraints ensuring that the demand for each month is satisfied by the chosen potential locations.This grid model can be solved using the Simplex algorithm to find a solution which might violate the Integrality constraint on the variables, and is then combined with a branch and bound algorithm to search for integer values. Solving the grid model with an ILP solver proved to be the best approach, and we will use it for the two-stage optimization model.

6 Two-stage optimization model

We now extend the grid model (Section 4), to include forecast data, namely the costs of potential parks and expected electricity demand for 2020. The impact of considering the two-stage model is illustrated in the example below. Let a and b be 2 potential renewable parks. Assume a monthly demand with an annual growth rate, and assume that each park can satisfy alone the present demand but not the forecast one. We have the following combinations of scenarios shown in Table (1), of which 3 satisfy all the constraints and the optimal one shall be the one with lowest cost that relies on forecast cost reductions (either scenario 8 or 9 depending on the cost values).

	Build park a	Build park b	Demand 2010	Demand 2020
1	no	no	no	no
2	no	yes/2010	satisfied	no
3	no	yes/2020	no	no
4	yes/2010	no	satisfied	no
5	yes/2020	no	no	no
6	yes/2020	yes/2020	no	satisfied
7	yes/2010	yes/2010	satisfied	satisfied
8	**yes/2010**	**yes/2020**	**satisfied**	**satisfied**
9	**yes/2020**	**yes/2010**	**satisfied**	**satisfied**

TABLE 1 – All possible scenarios of building a and b

The uncertainty lies in the technologies costs, as well as the forecast electricity demand for 2020. It can be estimated with existing forecasting curves of market and demographic studies. It is expected as shown in Figure 1, that the cost for renewable energy technologies will decrease in the coming years [2]. For each type of uncertainty, the current approach considers an annual growth rate for the electricity demand (that can be tuned by the end-user), and a total future cost per potential park according to the technology used and its transportation cost. Thus the initial grid model is extended with the following underlined terms and equations. The objective now is to find the minimum total cost for satisfying the demand of today and tomorrow, by taking into account forecast values.

The input for the final model is :

$$C_i = \textit{Cost of park } i \tag{16}$$
$$\underline{FC_i} = \underline{\textit{Future cost of the } i^{th} \textit{ location}} \tag{17}$$
$$M_i = \textit{Maximum area of park } i \tag{18}$$
$$(x_i, y_i) = \textit{Coordinates of park } i \tag{19}$$
$$G_{ij} = \textit{Watts/m}^2 \textit{ produced by park } i \textit{ in month } j \tag{20}$$
$$D_j = \textit{Electricity demand in 2010,month } j \tag{21}$$
$$\underline{r} = \underline{\textit{Annual growth rate of electricity demand}} \tag{22}$$
$$\underline{FD_j} = \underline{D_j \times r \textit{ future demand, month j, 2020}} \tag{23}$$

Model Variables We extend the set of variables from the short-term model, with new ones as shown below.

$$\mathbf{X_i} = \begin{cases} 1 & \textit{The } i^{th} \textit{ location is to be built in the present} \\ 0 & \textit{otherwise} \end{cases} \tag{24}$$

$$\underline{\mathbf{FX_i}} = \begin{cases} 1 & \textit{The } i^{th} \textit{ location is to be built in the future} \\ 0 & \textit{otherwise} \end{cases} \tag{25}$$

Constraints The integer linear program has the following linear constraints :

$$\sum_{i=1}^{n} G_{ij} \cdot \mathbf{X_i} \geq D_j, \ \forall j \in \{1, .., m\} \tag{26}$$
$$\underline{\sum_{i=1}^{n} G_{ij} \cdot (\mathbf{X_i} + \mathbf{FX_i}) \geq FD_j, \ \forall j \in \{1, .., m\}} \tag{27}$$
$$\underline{0 \leq \mathbf{X_i} + \mathbf{FX_i} \leq 1, \ \forall i \in \{1, .., n\}} \tag{28}$$
$$\mathbf{X_i} \in \{0, 1\}, \quad \mathbf{FX_i} \in \{0, 1\}, \ \forall i \in \{1, .., n\} \tag{29}$$

Constraints 26 ensure that the locations chosen to be built in the present satisfy the current demand, while constraints 27 ensure that all chosen locations built in the present and the future satisfy the future demand as well. Constraints 28 prevent each potential location from being built twice.

Objective Function The objective function defined in the approach is obviously a linear one. Since we seek to minimize the sum of all costs, given by the expression :

$$Cost = \sum_{i=1}^{n} \mathbf{X_i} \cdot C_i + \sum_{i=1}^{n} \mathbf{FX_i} \cdot FC_i \tag{30}$$

The cost expression is the sum of two scalar products, one for the locations chosen to be built today, and the other for the locations chosen to be built in the future. This model is also solved using the Simplex algorithm, with built-in branch and bound to search for an integer optimum.

7 Implementation

We now discuss the implementation of the different algorithms used and our prototype. The entire process of implementing all models was done on an Ubuntu 10.04 (Linux-based) Operating System.We first highlight the sources of the data used during testing and evaluation. The DP algorithm (14) as well as the constrained local search one (5.2), were implemented in `C++`. The DP algorithm uses top-down DP. The ECL^{i}PS^e [23] platform was chosen to implement the ILP model thanks to its high level of abstraction and hybridization libraries with the `cplex` solver.

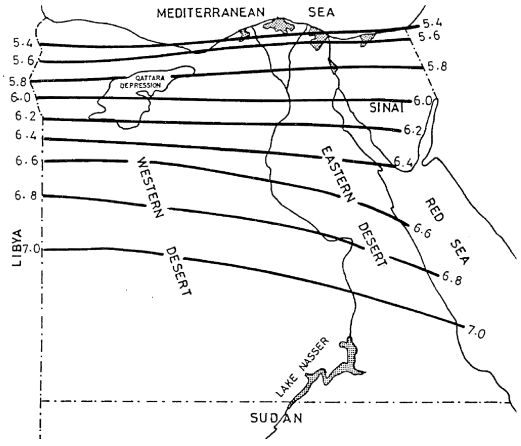


FIGURE 2 – Solar radiation map for March

Data pre-processing We purchased from a governmental body the wind and solar solar atlas of Egypt [16, 8]. In the form of printed documents and maps, they contain detailed information about solar radiation, wind speed, wind directions for each month of the year for a given location (resulting from analysis over the past 8 years). There are 12 maps in the solar atlas, each specifying an amount of watts/m^2 for each month for a given latitude. In the solar atlas, the annual average rate of electricity is given in a single map. A sample of the solar map for one month is shown in Figure 2. The information in these maps was extracted using `Python`'s image processing library `SciPy` [19]. We scanned the atlases, then processed the maps' images by labeling the different regions over each map with their corresponding solar radiation or wind power [7]. The result of the extraction process was the generation of a function that takes as input (x, y, t) and returns the kWh value, representing the gained rate of electricity at location (x, y) during month t.

Dataset Generation A dataset is composed of n points chosen and marked on a map like Figure 2. The gain of these locations is derived from the gain matrix G for the dataset. The costs for these locations is predefined according to the type of park considered (wind, solar).

Graphical User Interface The GUI module was implemented using `Java`. In order to enable the user to freely select locations from the map of Egypt, the OpenStreetMap [10] library was used. The user inputs data regarding the cost and maximum area for each location selected on the map. The database contains default data for the current demand of Egypt as well as the forecasted cost data shown in Figure 1. This information is stored and available for the solver to read during execution.

8 Evaluation

We first present the results of evaluating the scalability of the different algorithms we implemented : DP, DP with preprocessing using sorting (DPS), constrained Local search (LS), and ILP. We show the results on the average running time of 20 instances of the problem chosen randomly. This average is taken over all instances of the same problem size, n.

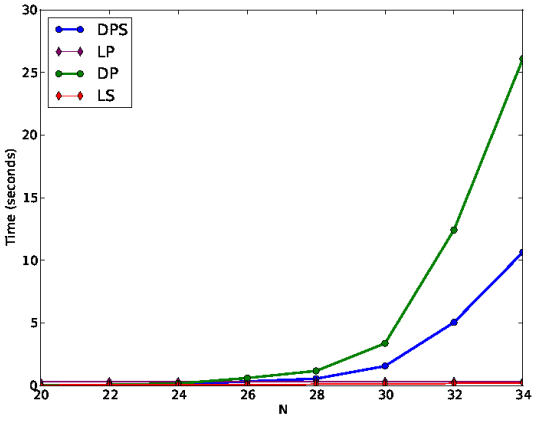


FIGURE 3 – Runtime of DP, DPS, ILP, and LS approaches for $20 \leq n \leq 34$

In Figure 3 we show the CPU running times of all

approaches for a group of datasets where $20 \leq n \leq 34$. The approaches that proved efficient for this problem size are LS and ILP. This is expected, since DP and DPS have a pseudo-polynomial running time.

Now we show the performance of the LS and ILP algorithms for larger values of n, $40 \leq n \leq 220$, see Figure 4. The results show a polynomial increase in the running time with respect to n. This reflects the $O(n^3)$ time complexity of the LS approach used. On the other hand, the ILP approach yet proves to be extremely fast and robust in solving all instances.

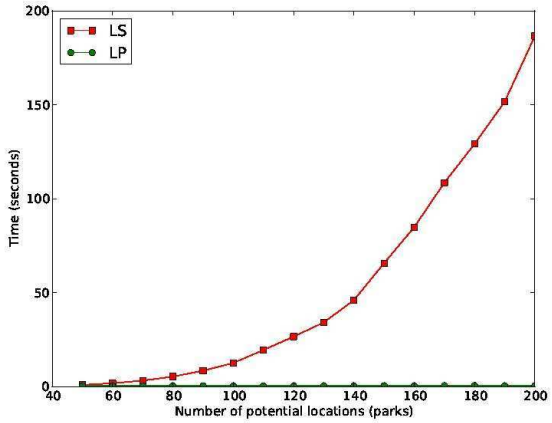


FIGURE 4 – Runtime of ILP and LS approaches for $40 \leq n \leq 220$

Forecast ILP Model The solution was broken into two stages. First the simple gridmodel described in Section 4 is solved. The two-stage model builds on this first solution by constraining the total cost to be smaller than the one produced by the grid model. It becomes an upper bound to the extended cost function.

We show that adding this constraint and solving the two-stage problem with the constrained new objective function (6), even though it requires a second run of the Cplex solver, it greatly speeds up the runtime of the entire algorithm. In other words, solving the forecast model without the upper bound constraint on the cost is less efficient than solving the grid model, acquiring the upper bound, then solving the forecast model, as shown in Figure 5. The testing was done for larger number of potential locations, $50 \leq N \leq 250$ given its efficiency.

Solution Quality We now illustrate the impact of reasoning with forecast demand and cost values in terms of the solution quality and difference in final

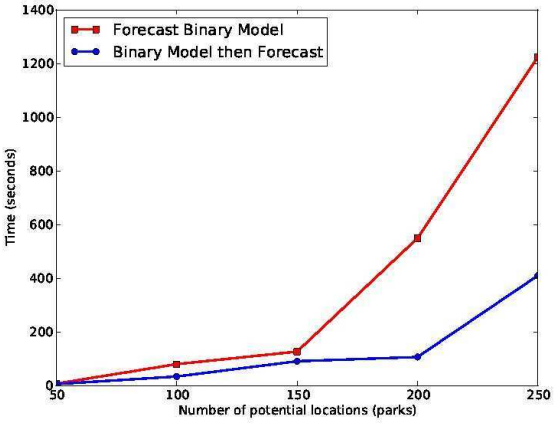


FIGURE 5 – One stage versus two-stage execution with forecast data

parks choice. A set of 10 potential solar and wind parks are initially placed on the GUI. These two sets are chosen by randomly selecting coordinates on the map of Egypt.The output of the algorithm is the optimal set of parks to be built in the present or in 10 years time in order to satisfy both demands. In Figure 6, the optimal set is shown when only current data is considered. While in Figure 7, it is shown how the optimal solution differs when we account for forecast costs and demands. We can see that solar park 8 and wind parks 1 and 7 are best to be built in the future and wind park 5,7 and 9 for instance should not be considered at all when future data is taken into account.

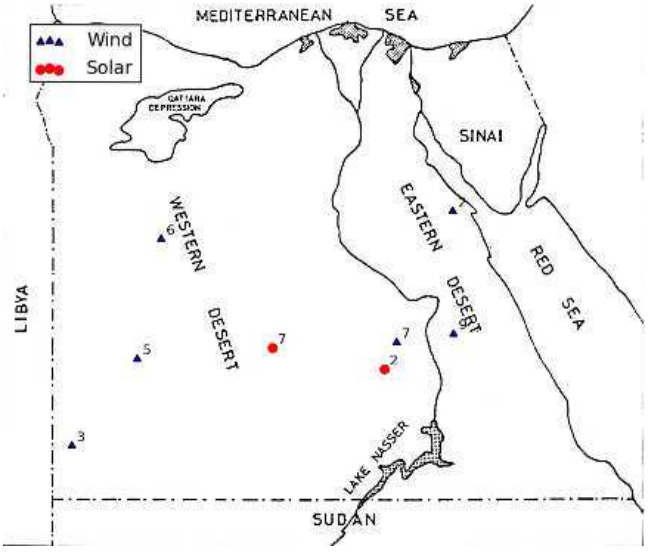


FIGURE 6 – Chosen locations using grid model

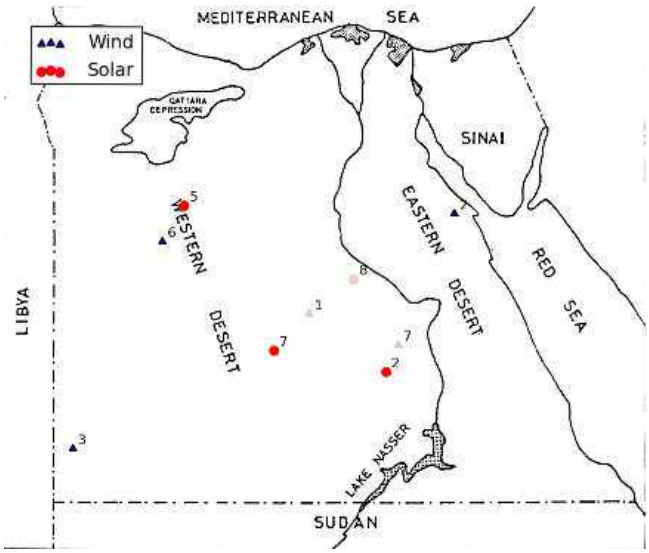


FIGURE 7 – Chosen locations with two-stage model

9 Conclusion

In this paper we have extended the state-of-the art models for optimal park selection, to account for both current and forecast data in the decision making process. We showed how this impacts greatly the solution quality, and also how it can be achieved efficiently using a two-stage approach as an ILP model. Future work includes a study of different models to represent the forecast data such as CDF-intervals [18] and probabilistic approaches. Strong interests from domain experts indicates the potentials of such a tool.

Références

[1] A.N. Arnette. *A Spatial Decision Support System for the Development of Multi-Source Renewable Energy Systems*. PhD thesis, Virginia Polytechnic Institute and State University, 2010.

[2] S.R. Bull. Renewable energy today and tomorrow. *Proceedings of the IEEE*, 89(8) :1216–1226, 2001.

[3] S.R. Bull. The integration of renewables. 2004.

[4] S. Dreyfus. Richard bellman on the birth of dynamic programming. *Operations Research*, pages 48–51, 2002.

[5] Energy efficiency ad Renewable energy. www.eere.energy.gov/wind.

[6] M.N. El-Kordy, M.A. Badr, Abed K.A., and Ibrahim S.M.A. Economical evaluation of electricity generation considering externalities. *Renewable Energy*, 25 :317–328.

[7] Saher El-Neklawy. Extracting solar radiation data from egyptian solar atlas. Technical report, GUC, August 2011.

[8] J. C. Hansen et al. *Wind Atlas For Egypt*. NREA, 2005.

[9] D. Ghosh, N. Chakravarti, SS Fatima, M. Wooldridge, and NR Jennings. A competitive local search heuristic for the subset sum problem. *Computers and Operations Research*, 26(3) :271–280, 1999.

[10] M. Haklay and P. Weber. Openstreetmap : User-generated street maps. *Pervasive Computing, IEEE*, 7(4) :12–18, 2008.

[11] G. Heal. The economics of renewable energy. *National Bureau of Economic Research*, 2009.

[12] R. King, H. Rughooputh, and K. Deb. Evolutionary multi-objective environmental/economic dispatch : Stochastic versus deterministic approaches. In *Evolutionary Multi-Criterion Optimization*, pages 677–691. Springer, 2005.

[13] J.M. Loiter and Norberg-Bohm V. Technology policy ad renewable energy :public roles in the development of new energy technologies. *Energy Policy*, 27 :85–97, 1999.

[14] Silvano Martello and Paolo Toth. Knapsack problems : algorithms and computer implementations. 1990.

[15] T. Nakata, K. Kubo, and A. Lamont. Design for renewable energy systems with application to rural areas in japan. *Energy Policy*, 33(2) :209–219, 2005.

[16] New and Renewable Energy Authority. *Egyptian Solar Radiation Atlas*. NREA, 1998.

[17] S.D. Pohekar and Ramachandran M. Application of multi-criteria decision making to sustainable energy planning - A Review. *Renewable and Sustainable Energy Reviews*, 8(4) :365–381, 2004.

[18] A. Saad, C. Gervet, and S. Abdelnnadher. Constraint Reasoning with Uncertain Data using CDF-Intervals. *In proceedings of CP-AIOR'10*, pages 292–306, 2010.

[19] SciPy – Scientific Tools for Python. http ://www.scipy.org/.

[20] K. Subramanyan, U.M. Diwekar, and A. Goyal. Multi-objective optimization for hybrid fuel cells power system under uncertainty. *Journal of power sources*, 132(1-2) :99–112, 2004.

[21] Earth Trends. Energy and resources. *The environmental information portal*, 2003.

[22] P. Van Hentenryck and L. Michel. *Constraint-Based Local Search*. The MIT Press, 2005.

[23] M. Wallace, S. Novello, and J. Schimpf. Eclipse : A platform for constraint logic programming. *ICL Systems Journal*, 12(1) :159–200, 1997.

[24] L. Wang and C. Singh. Environmental/economic power dispatch using a fuzzified multi-objective particle swarm optimization algorithm. *Electric Power Systems Research*, 77(12) :1654–1664, 2007.

Résolution parallèle de SAT : mieux collaborer pour aller plus loin

Gilles Audemard, Benoît Hoessen, Saïd Jabbour, Jean-Marie Lagniez et Cédric Piette *

Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens

{audemard,hoessen,jabbour,lagniez,piette}@cril.fr

Résumé

La gestion de la base de clauses apprises est connue pour être une tâche ardue au sein des solveurs SAT. Dans le cadre des solveurs parallèles de type *portfolio*, la collaboration réalisée entre les différents processus au moyen de l'échange de clauses rend cette gestion plus complexe encore. Différentes techniques ont été proposées ces dernières années, mais les résultats pratiques restent en faveur de solveurs ayant une collaboration limitée voire nulle. Ceci est principalement dû au fait que chacun des processus doit gérer le grand nombre de clauses générées par les autres. Dans cet article, nous proposons une nouvelle technique efficace pour l'échange de clauses au sein d'un solveur parallèle. Contrairement aux méthodes actuelles, notre approche repose sur des politiques pour l'exportation et pour l'importation de clauses, tout en utilisant des techniques récentes, qui ont prouvées leur efficacité dans le cadre séquentiel. Un grand nombre d'expérimentations tend à montrer l'intérêt des idées proposées.

1 Introduction

Depuis quelques années maintenant, la résolution pratique du problème SAT est un domaine de recherche des plus actifs. Avec l'avènement des architectures multicœurs, ces recherches se concentrent plus particulièrement sur le développement de démonstrateurs parallèles, capables de traiter des problèmes applicatifs de grande taille. Parmi les solveurs parallèles disponibles, citons `ManySat` [10], `SArTagnan` [12], `Plingeling` [4], `ppfolio` [16], `part-tree-learn` [11].

*Supporté par le CNRS et OSEO, avec le projet ISI "Pajero".

Il existe principalement deux types d'approches pour résoudre SAT en parallèle : les approches de type « diviser pour régner » et les approches de type *portfolio*. Ces dernières, qui consistent à faire collaborer différentes recherches concurrentes, sont à l'heure actuelle les plus efficaces en pratique. Dans le cas particulier du problème SAT, la collaboration entre les différentes recherches est principalement réalisée par le biais d'échanges d'informations (souvent sous la forme de clauses apprises). Il est cependant extrêmement difficile de prédire quelles clauses sont les plus pertinentes à partager. Afin de palier ce problème, `ManySAT` limite l'échange de clauses en ne partageant que celles possédant une taille inférieure à une certaine limite [9]. Toutefois, malgré le fait que cette approche ait obtenu plusieurs distinctions lors des précédentes compétitions, la plupart des solveurs actuels (`Plingeling` [4], `SArTagnan` [12], etc.) se limitent au partage des clauses unitaires. En outre, lors de la dernière compétition, un nouveau solveur de type *portfolio*, nommé `ppfolio`, a obtenu de très bon résultats, glanant un grand nombre de récompenses : `ppfolio` est en fait un simple script qui exécute de manière indépendante différents solveurs séquentiels parmi les plus performants. Par conséquent, il semble que ne pas faire collaborer (ou très peu) soit un choix tout à fait acceptable d'un point de vue empirique. Ce constat suggère que les stratégies utilisées actuellement pour gérer l'échange d'informations entre les recherches concurrentes ne sont pas matures, et semblent pouvoir être améliorées.

Prédire l'utilité (future) d'une clause est également un problème connu pour la résolution de SAT dans le cadre séquentiel. En effet, les solveurs SAT modernes sont capables d'apprendre une clause après chaque conflit, et l'ajout de ces dernières a une conséquence directe sur l'efficacité du solveur, puisqu'elles influent sur les performances de la

propagation unitaire. Afin de palier ce problème, la base de clauses apprises est périodiquement purgée de certaines clauses jugées inutiles pour la suite de la recherche. Récemment, une nouvelle mesure nommée *psm* [1] a été proposée afin de gérer la base de clauses apprises. Cette mesure consiste à comparer l'interprétation (partielle) courante à l'ensemble des littéraux de chaque clause apprise. L'idée générale de cette approche est la suivante : si l'intersection entre l'interprétation courante et les littéraux d'une clause est grande, alors la clause sera probablement inutile dans le sous-espace de recherche dans lequel le solveur se trouve. Si au contraire le nombre de littéraux de l'intersection est petit, alors la clause a de fortes chances d'être utilisée dans le processus de propagation unitaire, et par conséquent de réduire l'espace de recherche. Cette nouvelle mesure a permis la mise en place d'une nouvelle stratégie de gestion de la base de clauses apprises. Cette dernière, contrairement aux autres politiques, permet de geler une clause lorsque celle-ci est considérée comme inutile dans l'espace de recherche courant. Ainsi, périodiquement les clauses apprises sont évaluées à l'aide de la mesure *psm*, afin d'en activer certaines et d'en geler d'autres. Différentes expérimentations ont permis de montrer que cette approche a un très bon comportement en pratique, et permet très souvent de sélectionner les clauses qui sont utiles dans un futur proche.

Dans ce papier, nous étendons les résultats présentés dans [1] à la résolution parallèle de SAT. Nous proposons différentes politiques d'exportation et d'importation de clauses, pour les différents *threads* d'un solveur *portfolio*. Cet article est structuré ainsi : la section suivante présente les notations et notions nécessaires à sa lecture, puis la section 3 présente quelques résultats préliminaires sur l'échange de clauses au sein d'un solveur parallèle de type *portfolio*. Ensuite, dans la section 4, nous présentons notre approche nommée `PeneLoPe`, puis étudions expérimentalement son comportement vis-à-vis des autres solveurs de l'état de l'art dans la section 5. Enfin, nous concluons en fournissant quelques perspectives.

2 Pré-Requis

Nous supposons le lecteur familier avec les notions liées au problème de satisfiabilité d'une formule propositionnelle (ie. variable x, littéral positif x ou négatif $\neg x$, clause, clause unitaire, interprétation et CNF). Nous rappelons juste brièvement le schéma général d'un solveur CDCL (*Conflict-Driven, Clause Learning*). Une branche d'un solveur CDCL est une séquence de décisions, suivies de propagations des littéraux unitaires, répétées jusqu'à ce qu'un conflit survienne. Chaque variable de décision choisie au moyen d'une heuristique, généralement basée sur l'activité, est affectée à un niveau de décision donné, les littéraux propagés en conséquence ayant le même niveau de décision. Chaque fois qu'un conflit survient, un *nogood* est calculé avec une méthode donnée, généralement celle nommée FUIP (*First Unique Implication Point*) [14, 19]. Le *nogood* est ajouté à la base des clauses apprises et un saut arrière est effectué. En outre, des redémarrages sont effectués périodiquement. Le lecteur désirant plus de détails sur les solveurs de type CDCL peut se référer à [7]. Dans la suite, certaines notions nécessaires à la compréhension du papier sont détaillées.

2.1 Gestion de la base de clauses apprises

La taille de la base de clauses apprises influe énormément sur les performances d'un solveur. En effet, conserver trop de clauses peut nuire à l'efficacité du processus de propagation unitaire (celui-ci étant intrinsèquement lié au nombre de clauses), tandis que supprimer trop de clauses peut faire perdre le bénéfice de l'apprentissage. Afin de palier ce problème, les solveurs effectuent périodiquement un nettoyage de la base qui consiste à supprimer les clauses apprises considérées comme inutiles. Par conséquent, identifier ce qu'est une bonne clause (ie. utile à l'établissement de la preuve) est clairement un challenge important. La première mesure de qualité proposée, certainement la plus populaire, est basée sur la notion d'activité inhérent à l'heuristique de choix de variable VSIDS. Plus précisément, une clause apprise est considérée comme utile à la preuve si elle apparaît souvent comme *raison* d'un conflit. Typiquement, cette stratégie de suppression considère qu'une clause utile dans le passé le sera dans le futur. Dans [2], une autre mesure, appelée *lbd*, est utilisée afin d'estimer la pertinence d'une clause. Cette nouvelle mesure est basée sur le nombre de niveaux de décision différents apparaissant dans une clause apprise lorsque que cette dernière est générée. Des expérimentations ont permis de mettre en exergue le fait que des clauses possédant une valeur de *lbd* faible sont plus souvent utilisées que celles avec une valeur de *lbd* élevée. Toutefois, bien que ces mesures soient efficaces en pratique, elles restent heuristiques et par conséquent ne peuvent éviter la suppression de clause utiles pour la suite de la recherche.

2.2 Solveur SAT parallèle

Il existe deux types d'approches pour résoudre SAT en parallèle. D'une part, les approches de type « diviser pour régner » consistent à diviser le problème en plusieurs sous-problèmes qui sont ensuite résolus indépendamment par différentes unités de calcul [5, 6]. L'un des inconvénients de ce modèle réside dans le partage de la charge de travail entre les processeurs. En effet, puisque la taille des sous-problèmes n'est pas connue à l'avance, il est nécessaire de mettre en place une politique de rééquilibrage entre les différentes unités de calcul. Ce rééquilibrage peut être fait de manière statique ou dynamique [11]. Notons que certaines

approches sont aussi capables de transférer des informations (le plus souvent sous la forme de *nogood*) entre les différentes unités de calcul [11, 17].

D'autre part, les approches de type *portfolio* mettent plusieurs solveurs en concurrence afin de résoudre le problème. Ces méthodes permettent ainsi d'exploiter la complémentarité entre différentes stratégies utilisées par les solveurs CDCL [10, 4, 12]. Puisque chaque processeur travaille avec la formule initiale, ces approches ne nécessitent pas la mise en place de stratégies d'équilibrage de charge, et la coopération est assurée par l'échange de clauses apprises par les différentes recherches. Malgré sa simplicité, ce type d'approche n'est pas toujours évident à mettre en place. En effet, le choix des stratégies utilisées est prépondérant (particulièrement si le nombre de processeurs disponibles est faible). De manière générale, l'objectif est de couvrir le plus de problèmes possibles.

Comme nous l'avons déjà souligné précédemment, la taille de la base de clauses apprises est un critère crucial pour l'efficacité des solveurs séquentiels. Par conséquent, dans le cadre d'une approche de type *portfolio*, il n'est pas souhaitable de partager systématiquement toutes les clauses apprises par les différentes recherches, au risque de voir la taille de la base de clauses apprises augmenter trop rapidement. Afin de conserver le coté collaboratif de ces approches, il est donc nécessaire de définir des critères afin d'identifier les clauses qui peuvent être échangées. Une manière naturelle de gérer cela consiste à considérer la taille des clauses comme critère de sélection. Les auteurs de `ManySAT` [10] proposent par exemple de ne partager que les clauses dont la taille est inférieure à 8. Cependant, après avoir observé que les clauses de petite taille apparaissent de moins en moins souvent durant la recherche, ils ont ensuite proposé de régler dynamiquement la taille limite des clauses échangées entre les différents *threads* [9]. Toutefois, il est surprenant qu'un solveur tel que `Plingeling` qui n'échange entre les *threads* que les clauses unitaires soit l'un des vainqueurs de la compétition SAT 2011. Plus surprenant encore : le solveur `ppfolio`, qui a obtenu un nombre important de récompenses lors de cette même compétition, n'échange aucune information et ne fait qu'exécuter en parallèle -et de manière indépendante- plusieurs solveurs séquentiels. Ces dernières observations montrent à quel point les techniques liées à la collaboration entre les recherches semblent pouvoir être améliorées.

3 Expérimentations préliminaires

Dans un premier temps, nous avons conduit des expérimentations préliminaires afin d'étudier le comportement des solveurs *portfolio* vis-à-vis des clauses échangées. Pour un solveur séquentiel, une « *bonne* » clause apprise est une clause utilisée dans le processus de propagation unitaire et l'analyse de conflits. Pour les solveurs *portfolio*, on peut se baser naturellement sur la même idée : une « *bonne* » clause partagée est une clause qui aidera au moins un autre *thread* à réduire son espace de recherche, c'est à dire, qui participera à la propagation. Nous avons donc voulu connaître l'utilité des clauses partagées dans les solveurs *portfolio*.

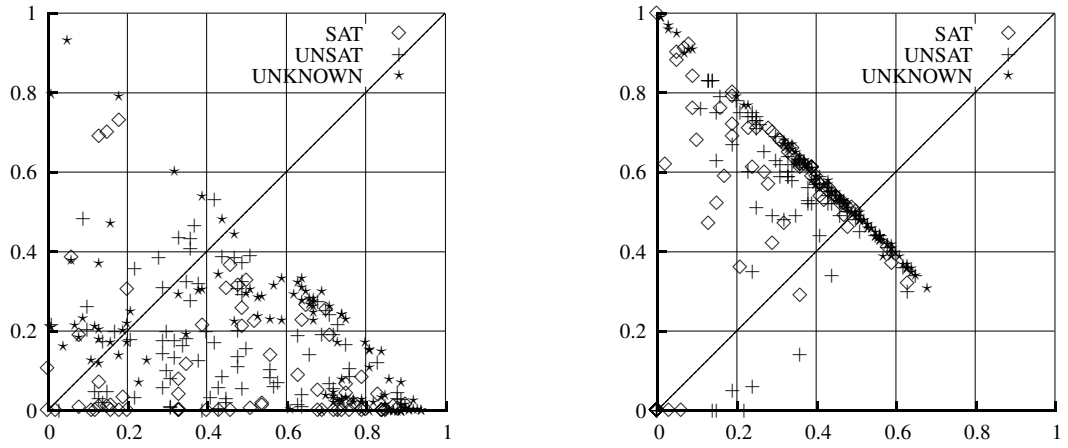
Pour cela, nous avons mené des expérimentations [1] sur `ManySAT 2.0` (basé sur `Minisat 2.2`), l'un des solveurs *portfolio* les plus efficaces, et au sein duquel une vraie collaboration est réalisée. Ce solveur ne différencie les *threads* que sur les premières variables de décision, qui sont choisies au hasard. Ainsi, exceptée l'interprétation initiale, chaque *thread* associé à un solveur CDCL possède exactement le même comportement (en terme de redémarrages, heuristique de choix de variable, etc.), ce qui permet de faire une comparaison plus juste, limitant les effets de bord liés aux autres paramètres. Par défaut, toutes les clauses apprises de taille inférieure à 8 sont partagées entre les *threads*. De plus, `ManySAT` possède un mode déterministe [8] que nous avons utilisé afin de rendre reproductibles les différents résultats proposés dans cet article [2].

Soit $\mathcal{SC}$ l'ensemble des clauses partagées, c'est-à-dire l'union de toutes les clauses exportées par un *thread* vers les autres. Nous avons considéré deux types de clauses partagées. Tout d'abord, les clauses qui ont été utilisées (au moins une fois) par un *thread* dans le processus de propagation unitaire. Nous notons cet ensemble de clauses $used(\mathcal{SC})$. L'autre type de clauses considéré est celui des clauses qui ont été supprimées sans avoir du tout participé à la recherche. Nous le notons $unused(\mathcal{SC})$. Clairement, l'ensemble $\mathcal{SC} \backslash (used(\mathcal{SC}) \cup unused(\mathcal{SC}))$ représente les clauses qui, à la fois, n'ont pas encore été utilisées ni supprimées.

La Figure 1 illustre les résultats obtenus par les différentes instances. Ils y sont reportés de la manière suivante : chaque point de la figure correspond à une instance, l'axe des x représentant le taux d'utilisation des clauses partagées ($\#used(\mathcal{SC})/\#\mathcal{SC}$) tandis que l'axe des y représente celui des clauses inutiles et supprimées ($\#unused(\mathcal{SC})/\#\mathcal{SC}$) et nous reportons la moyenne sur les différents *threads*. La Figure 1(a) donne les résultats pour `ManySAT`. Nous pouvons tout d'abord remarquer que le taux d'utilisation des clauses partagées diffère fortement d'une instance à l'autre. Nous pouvons également remarquer que dans de nombreux cas, `ManySAT` conserve

1. Toutes les expérimentations conduites dans cet article sont réalisées sur des bi-processeurs `Intel XEON X5550` 4 cœurs à 2.66 GHz avec 8Mo de cache et 32Go de RAM, sous Linux CentOS 6. (kernel 2.6.32). Chaque solveur utilise 8 *threads*. Le temps limite alloué pour résoudre une instance est de 1200 secondes WC. Nous avons utilisé les 300 instances de la catégorie *application* de la compétition SAT 2011.

2. Toutes les traces obtenues, ainsi que différentes statistiques, sont disponibles à `http://www.cril.fr/~hoessen/penelope.html`



(a) `ManySAT` (b) `ManySAT` aggressif

FIGURE 1 – Comparaison entre les clauses partagées utilisées et les clauses partagées non utilisées et supprimées. Chaque point correspond à une instance. L'axe des x donne le taux d'utilisation des clauses partagées $\frac{\#used(\mathcal{SC})}{\#\mathcal{SC}}$, alors que l'axe des y donne celui des clauses partagées non utilisées et supprimées $\frac{\#unused(\mathcal{SC})}{\#\mathcal{SC}}$.

des clauses pendant toute la recherche (points proches de l'axe des x). Ceci est dû à la politique non agressive de `Minisat` où dans de nombreux cas, très peu de nettoyages de clauses sont effectués. Les différents *threads* peuvent donc conserver les clauses inutiles durant une longue période, causant un sur-coût sans engendrer de bénéfice.

Nous avons réalisé les mêmes expérimentations en utilisant une politique de nettoyage des clauses beaucoup plus agressive, a savoir celle utilisée dans [1] (voir la section 4) et nous reportons les résultats dans la Figure 1(b). Ici, pour de nombreuses instances, les clauses sont supprimées avant d'avoir été utilisées et le taux de clauses utiles décroit fortement par rapport à la version basique de `ManySAT`.

Ces résultats s'expliquent facilement. Si peu de nettoyages sont effectués, les différents solveurs doivent supporter de nombreuses clauses, qu'elles se révèlent utiles ou non. Cela fournit donc beaucoup d'informations et permet donc de propager plus de littéraux. Mais en contrepartie, les différents *threads* doivent alors maintenir les structures de données sur de nombreuses clauses inutiles, ralentissant ainsi le processus de recherche. Dans le cas contraire, si beaucoup de nettoyages sont réalisés, un autre problème survient. Si une clause partagée ne participe pas directement au processus de propagation et d'analyse de conflit, il y a de fortes chances qu'elle soit rapidement supprimée. Ainsi, les *threads* perdent beaucoup de temps à importer ces clauses, puis à les supprimer peu après.

Nous pouvons également noter que l'utilisation du *lbd* pour mesurer la qualité des clauses apprises ne semble pas non plus adéquate. En effet, les clauses partagées sont de petites clauses et ont donc une petite valeur de *lbd*. Les clauses importées ont alors plus de chance d'être préférées aux clauses générées par le *thread* lui-même. Ainsi, même s'il semble possible de paramétrer plus précisément la stratégie de nettoyage afin d'obtenir un solveur plus robuste, nous pensons que la gestion actuelle des clauses apprises n'est pas appropriée dans le cas du partage de clauses des solveurs *portfolio*. Nous proposons donc une nouvelle stratégie dans la prochaine section.

4 Sélectionner, partager et activer les bonnes clauses

La gestion des clauses apprises est connue pour être un problème difficile dans le cas séquentiel. Gérer celles provenant d'autres fils d'exécution conduit nécessairement à de nouvelles difficultés :

- Les clauses importées peuvent être sous-sommées par des clauses déjà présentes. La sous-sommation étant une opération gourmande en temps, il est nécessaire d'offrir la possibilité de supprimer périodiquement certaines clauses apprises.
- Une clause importée peut être inutile durant une longue période avant d'être utilisée dans la propagation.
- Chaque fil d'exécution doit gérer un plus grand nombre de clauses
- Il est très difficile de caractériser ce qu'est une bonne clause importée.

Pour chacune de ces raisons, nous proposons d'utiliser la politique de gestion dynamique des clauses apprises proposée par [1] au sein de chaque fil d'exécution. Cette technique récente permet d'activer ou de geler certaines clauses apprises, importées ou générées localement. L'avantage de cette technique est double. Étant donné qu'elles peuvent

être gelées, la surcharge calculatoire occasionnée par les clauses importées est grandement réduite. De plus, les clauses importées, qui peuvent se révéler utiles dans un futur plus ou moins proche de la recherche, sont activées au moment adéquat. La prochaine section présente de manière plus précise cette méthode.

Geler certaines clauses

La stratégie proposée dans [1] est très différente de celles proposées par le passé (voir section 2). Elle est en effet basée sur le gel et l'activation dynamique des clauses apprises. À un certain point de la recherche, les clauses les plus prometteuses sont activées tandis que les autres sont gelées. De cette façon, les clauses apprises peuvent être écartées pour les prochaines propagations, mais peuvent par la suite être réactivées. Cette stratégie ne peut être utilisée avec les mesures d'activité basées sur le VSIDS ou le *lbd*. En effet, la mesure d'activité (inspirée par VSIDS) est dynamique mais ne peut être utilisée que pour mettre à jour l'activité des clauses actives -c'est-à-dire participant effectivement à la recherche- , tandis que *lbd* est soit statique (et ne change donc pas durant la recherche), soit dynamique auquel cas on rencontre le même problème que la mesure basée sur VSIDS. De ce fait, cette stratégie est associée avec une autre mesure afin de mesurer l'utilité d'une clause apprise [1].

Soient c une clause apprise par le solveur et ω l'interprétation courante résultant de la dernière polarité des variables de décisions [15]. La valeur *psm* de la clause c par rapport à ω, notée $psm_\omega(c)$, est égale à $psm_\omega(c) = |\omega \cap c|$.

Basée sur l'interprétation courante, *psm* se révèle être une mesure hautement dynamique. Son but est de sélectionner le contexte approprié à l'état courant de la recherche. Dans cette optique, les clauses ayant une petite valeur de *psm* sont considérées comme utiles. En effet, de telles clauses ont plus de chance de participer à la recherche, par le mécanisme de propagation unitaire ou en étant falsifiées. Au contraire, les clauses avec une grande valeur de *psm*, ont une plus grande probabilité d'être satisfaites par deux littéraux ou plus, les rendant inutiles pour la recherche en cours.

Ansi, seules les clauses ayant une petite valeur de *psm* sont sélectionnées et utilisées par le solveur dans le but d'éviter un surcoût calculatoire lié à la maintenance des structures de données pour ces clauses inutiles. Néanmoins, une clause gelée n'est pas supprimée, mais est gardée en mémoire, puisqu'elle peut se révéler utile par la suite. Au fur et à mesure que l'interprétation courante évolue, l'ensemble de clauses utilisées évolue également. Dans cette optique, la mesure *psm* est calculée périodiquement et les ensembles de clauses gelées et actives sont ainsi mis à jour.

Soit P_k une suite où $P_0 = 500$ et $P_{i+1} = P_i + 500 + 100 \times i$. Une procédure nommée "*updateDB*" est appelée chaque fois que le nombre de conflit atteint P_i (où $i \in [0..\infty]$. Cette procédure calcule la nouvelle valeur *psm* de chaque clause apprise (gelée ou active). Une clause avec une valeur *psm* inférieure à une limite l est activée, dans le cas contraire, elle est gelée. De plus, une clause qui n'est pas activée après k itérations (par défaut, $k = 7$) est définitivement supprimée. De façon similaire, une clause qui reste active plus de k étapes sans participer à la recherche est également supprimée.

Grâce aux mesures *psm* et *lbd*, il est désormais possible de définir une politique pour l'échange de clauses. Dans un solveur CDCL typique, un *nogood* est appris après chaque conflit. Il est donc clair que toutes les clauses ne peuvent être partagées. De ce fait, lorsqu'il y a collaboration, ces clauses doivent être filtrées selon un critère. À notre connaissance, dans tous les solveurs *portfolio*, ce critère est uniquement basé sur les informations de l'expéditeur de la clause, le destinataire n'ayant pas d'autres choix que d'accepter une clause jugée utile par un autre solveur.

Nous présentons donc une technique où l'expéditeur *et* le destinataire peuvent avoir leur propres stratégies. Chaque expéditeur (stratégie d'export) essaie de trouver dans sa base de clauses apprises les informations les plus pertinentes pour aider les autres *threads*. De plus, le destinataire (stratégie d'import) peut ne pas accepter aveuglément les clauses qui lui sont envoyées. Nous avons nommé ce solveur prototype `PeneLoPe`[3] (**P**arallel **L**bd **P**sm solver).

Politique d'import de clauses Quand une clause est importée, différents cas peuvent être considérés en fonction du moment où la clause sera attachée pour que celle-ci participe à la recherche.

- *no-freeze* : chaque clause importée est directement activée, et sera évaluée (et potentiellement gelée) durant le prochain appel à *updateDB* .
- *freeze-all* : chaque clause importée est *gelée* par défaut, et ne sera utilisée par la suite que si elle remplit les conditions pour être activée.
- *freeze* : chaque clause est évaluée au moment de l'import. Elle est activée si elle est considérée comme prometteuse, selon les mêmes critères que si elle avait été générée localement.

Politique d'export de clause Puisque `PeneLoPe` est capable de geler certaines clauses, il semble possible d'en importer un plus grand nombre que dans le cas d'une gestion classique des clauses, où chacune d'elle est attachée jusqu'à sa possible suppression. De ce fait, nous proposons différentes stratégies, plus ou moins restrictives, pour sélectionner les clauses partagées :

3. En référence à la femme fidèle d'Ulysse, qui réalisait une tapisserie en liant de nombreux *fils* (*threads*)

psm	d'export	redémarrage	import	#SAT	#UNSAT	#SAT + #UNSAT
✓	*lbd limit*	*lbd*	*no freeze*	94	111	**205**
✓	*lbd limit*	*lbd*	*freeze*	89	**113**	202
✓	*size limit*	*lbd*	*freeze*	93	107	200
✓	*size limit*	*lbd*	*no freeze*	89	107	196
✓	*size limit*	*luby*	*no freeze*	**97**	98	195
✓	*lbd limit*	*lbd*	*freeze all*	89	102	191
✓	*size limit*	*luby*	*freeze all*	96	92	188
✓	*unlimited*	*lbd*	*freeze*	86	102	188
✓	*size limit*	*luby*	*freeze*	92	96	188
✓	*lbd limit*	*luby*	*freeze*	91	97	188
ManySAT	-	-	-	95	93	188
✓	*lbd limit*	*luby*	*no freeze*	90	94	184
✓	*unlimited*	*luby*	*freeze*	91	92	183
	size limit	*luby*	*no freeze*	92	90	182
✓	*unlimited*	*luby*	*no freeze*	89	88	177
✓	*size = 1*	*lbd*	*freeze*	89	88	177

TABLE 1 – Comparaison entre les différentes stratégies d'import, d'export & redémarrage, en utilisant le mode déterministe

- *unlimited* : chaque clause générée est exportée aux différents fils d'exécution.
- *size limit* : seules les clauses dont la taille est inférieure à une valeur donnée (8 dans nos expérimentations) sont exportées [10].
- *lbd limit* : une clause est exportée aux autres fils d'exécution si son *lbd* est inférieur à une limite donnée d (8 par défaut). Il est important de remarquer que la valeur du *lbd* peut être réduite au cours du temps. Ainsi, dès que $lbd(c)$ est inférieur à d, la clause est exportée.

Politique de redémarrage En plus des politiques d'échange, `PeneLoPe` permet également de choisir entre deux politiques de redémarrage.

- *Luby* : Soit l_n le n-ième terme de la série Luby [13]. Le n-ième redémarrage est effectué après $l_n \times \alpha$ conflits (α vaut 100 par défaut).
- *lbd* : Soit LBD_g la moyenne des valeurs de *lbd* de chaque clause apprise depuis le commencement de la recherche. Soit LBD_{100} la moyenne des 100 dernières clauses apprises. Cette politique induit qu'un redémarrage est effectué dès que $LBD_{100} \times \alpha > LBD_g$ (α vaut 0.7 par défaut) [2].

Différentes expérimentations ont été réalisées dans le but de comparer ces politiques d'import, d'export et de redémarrage. Différentes versions ont été exécutées et la table 1 présente une partie des résultats obtenus. Cette table rapporte pour chaque stratégie le nombre d'instances SAT (#SAT), UNSAT (#UNSAT) et total (#SAT + #UNSAT) résolues.

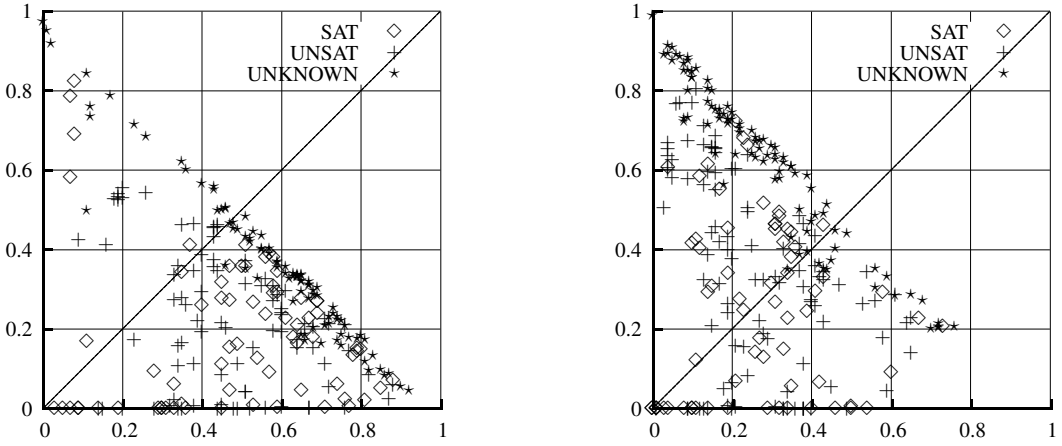
Comparons d'abord la stratégie d'import. Sans surprise, la politique *unlimited* obtient les plus mauvais résultats. En effet, aucune version utilisant cette stratégie n'est capable de résoudre plus de 190 instances. Chaque clause générée est exportée, et nous obtenons ainsi le niveau maximum de communication. Comme attendu, avec la multiplicité des fils d'exécution, les solveurs sont vite submergés par les clauses et leurs performances chutent.

C'est la raison pour laquelle des limites basées sur la taille ont été introduites avec l'idée que de petites clauses permettent de mieux filtrer l'espace de recherche et sont donc préférables. En effet, on constate dans la table 1 que la politique *size limit* surpasse *unlimited*. De plus, il est également possible de constater que l'échange des seules clauses unitaires n'offre pas d'aussi bonnes performances (*size = 1*), tout comme cela avait été annoncé préalablement. Il est également important de signaler que les clauses de grande taille peuvent grandement réduire la taille de la preuve [3].

L'utilisation de la valeur *lbd* d'une clause (*lbd*(c)) peut donc s'avérer bénéfique étant donné que *lbd*$(c) \leq$ taille(c). De ce fait, une politique utilisant *lbd* exporte plus de clauses que celle utilisant la taille (avec la même borne).

Ceci pourrait représenter un problème pour un solveur parallèle n'ayant pas la possibilité de geler certaines clauses. Cependant, puisque `PeneLoPe` contient un tel mécanisme, l'impact est grandement réduit. D'un point de vue empirique, la table 1 montre que *lbd limit* obtient les meilleurs résultats entre les différentes politiques.

Concentrons nous maintenant sur la politique de redémarrage. De façon évidente, la politique *luby* obtient globalement de moins bons résultats que celle utilisant *lbd*. Cela montre clairement l'intérêt de cette mesure introduite dans [2]. À propos de la stratégie d'import, aucune stratégie ne se révèle être clairement meilleure que les autres. En effet, le meilleur résultat relatif au nombre d'instances SAT est obtenu par *no freeze* (97) lorsqu'il est associée à la politique de redémarrage *luby* et *size limit* comme politique d'export, tandis que le meilleur résultat en nombre d'instance UNSAT est obtenu en utilisant la stratégie *freeze* (113). De plus, *no freeze* permet d'obtenir les meilleurs résultats globaux en résolvant 205 instances parmi les 300



(a) *size limit* + *luby* + *no freeze* (SLN)

(b) *lbd limit* + *lbd* + *freeze* (LLF)

FIGURE 2 – Comparaison entre les clauses importées révélées utiles et les clauses importées n'ayant pas été utilisées dans la propagation et supprimées. Chaque point représente une instance. L'axe x représente le taux de clauses utiles $\frac{\#used(\mathcal{SC})}{\#\mathcal{SC}}$, tandis que l'axe y représente le taux de clauses non utilisées et supprimées $\frac{\#unused(\mathcal{SC})}{\#\mathcal{SC}}$.

utilisées. Il serait ainsi audacieux de plaider envers l'une des 3 techniques proposées. Cependant, un grand nombre des politiques proposées obtiennent de meilleurs résultats que les techniques "classiques" d'échange de clauses, représentées dans la table 1 par `ManySat`.

Dans une seconde expérimentation, nous avons voulu évaluer le comportement de `PeneLoPe` avec certaines des politiques proposées. Dans ce but, nous avons reconduit le même type d'expérimentations que celle présentée dans la section 3 ; les résultats obtenus sont reportés dans la Figure 2. Dans un premier temps, nous avons essayé avec les politiques *size limit*, *luby*, et *no freeze* (dénoté `SLN`, voir Figure 2(a)).

Il est clairement visible que cette version se comporte bien, étant donné que la plupart de ces points sont situés sous la diagonale. De plus, pour la plupart des instances, $\frac{\#used\mathcal{SC}+\#unused(\mathcal{SC})}{\#\mathcal{SC}}$ est proche de 1 (nombreux points situés près de la seconde diagonale), ce qui indique que le solveur ne comporte que peu de clauses qui ne sont pas utilisées sans les avoir supprimées. La plupart d'entre elles sont donc utiles, tandis que les autres sont supprimées.

Ensuite, l'expérimentation fut reconduite en utilisant les politiques *lbd limit*, *lbd* , et *freeze* (dénoté `LLF`, voir Figure 2(b)). Au premier abord, le comportement de cette version est moins satisfaisant que la version `SLN`, du fait que pour la plupart des instances, plus de la moitié des clauses importées sont supprimées sans avoir été utiles dans la propagation. En réalité, dans cette version, un nombre bien plus important de clauses sont exportées du fait de la politique d'export *lbd limit*, ce qui conduit à un taux d'utilisation plus faible. Un réglage plus fin des paramètres (limite d'export pour les valeurs *lbd*, nombre de fois qu'une clause peut être gelée avant d'être supprimée, etc.) pourrait améliorer cela.

Si l'on regarde les statistiques détaillées de quelques instances, présentées dans la table 2, il apparaît que la version `LLF` partage en effet beaucoup plus de clauses que la version `SLN` (colonne nb_u). Notons aussi que cette table contient d'autres informations très intéressantes. Par exemple, il est possible de voir que pour certaines instances (par exemple `AProVE07-21`), quasiment 90% des clauses importées sont en fait gelées et ne participent pas immédiatement à la recherche, alors que pour d'autres instances la situation inverse se produit (`hwmcc...`). Ceci révèle la grande adaptabilité de la mesure *psm*.

D'un point de vue plus général, même si la politique *no-freeze* semble être meilleure en terme d'efficacité des communications entre fils d'exécution, elle possède l'inconvénient d'ajouter chaque clause importée à l'ensemble des clauses actives. Ceci implique que le nombre de propagation par seconde sera ralenti jusqu'au prochain ré-examen de la base de clauses apprises. Ceci peut être un problème si l'on augmente le nombre de fils d'exécution. D'un autre coté, la politique *freeze-all* ne ralentit pas le solveur, mais dans ce cas, il est possible que le solveur soit en train d'explorer un espace de recherche qui aurait pu être évité avec la politique *no-freeze*.

5 Comparaison avec les solveurs états de l'art

Dans cette section, nous proposons une comparaison de deux de nos prototypes avec les meilleurs solveurs SAT parallèles connus à ce jour. Nous avons ainsi sé-

Instance	Version	Temps	nb_c	nb_i	nb_f	nb_u	nb_d
dated-10-17-u	SLN	TO	1771	278 (0.15)	0%	45%	49%
	LLF	949	1047	1251 (0.83)	64%	20%	60%
hwmcc10-timeframe-	SLN	TO	5955	7989 (1.34)	0%	35%	60%
expansion-k50-eijkbs6669-tseitin	LLF	766	3360	15299 (4.55)	10%	11%	80%
velev-pipe-o-uns-1.1-6	SLN	150	981	69 (0.07)	0%	60%	24%
	LLF	48	296	173 (0.58)	41%	31%	33%
sokoban-sequential-p145-	SLN	TO	182	86 (0.47)	0%	92%	4%
microban-sequential.040	LLF	530	74	155 (2.09)	5%	58%	17%
AProVE07-21	SLN	10	78	83 (1.06)	0%	35%	16%
	LLF	31	143	506 (3.53)	89%	9%	57%
slp-synthesis-aes-bottom13	SLN	445	1628	194 (0.11)	0%	58%	30%
	LLF	91	309	298 (0.96)	71%	24%	49%
velev-vliw-uns-4.0-9-i1	SLN	TO	1664	262 (0.15)	0%	55%	40%
	LLF	906	1165	824 (0.70)	35%	37%	48%
x1mul.miter.shuffled-as.sat03-359	SLN	819	2073	421 (0.20)	0%	51%	37%
	LLF	280	680	1134 (1.66)	76%	16%	59%

TABLE 2 – Statistiques à propos d'instances UNSAT. Pour chacune d'entre elle et pour chaque version de PeneLoPe, nous reportons le temps WC nécessaire pour résoudre l'instance, le nombre de conflits (nb_c, en milliers), le nombre de clauses importées (nb_i, en milliers) avec entre parenthèses le taux entre nb_i et nb_c, le pourcentage de clauses gelée à l'import (nb_f), le pourcentage de clauses importées utiles (nb_u) et le pourcentage de clauses importées et supprimées sans avoir été utiles dans la recherche (nb_d). À l'exception du temps, chaque mesure est une moyenne sur les 8 fils d'exécution.

lectionné les solveurs qui se sont montrés les plus efficaces lors des dernières compétitions : ppfolio [16], cryptominisat [18], plingeling [4] et ManySat [10]. Pour PeneLoPe, nous avons choisi pour les deux versions la stratégie de redémarrage basée sur *lbd*, ainsi que la politique d'exportation *lbd limit*. Ces versions ne diffèrent donc que par leur politique d'importation de clauses, dont l'une est *freeze*, l'autre *no freeze*. Précisons également que contrairement à tous les résultats présentés précédemment, nous n'avons pas utilisé le mode déterministe dans cette partie, dans le but d'obtenir les meilleures performances possibles.

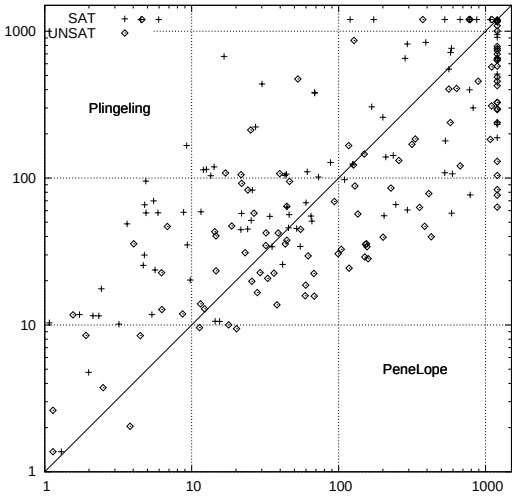
La Figure 3 donne les résultats selon différentes représentations. PeneLoPe dépasse en pratique tous les autres solveurs paralèlles. En effet, il parvient à résoudre 216 instances tandis qu'aucun autre solveur ne dépasse les 200 (Figure 3(a)). Notons cependant que si on ne considère que les instances satisfaisables, les meilleurs résultats sont obtenus par plingeling qui en résout 99. Ceci est particulièrement visible dans la Figure 3(b) où PeneLoPe et plingeling sont plus précisément comparés. Dans cette Figure, la plupart des points représentant des instances SAT sont assurément au dessus de la diagonale, illustrant la force de plingeling sur ce type de problèmes. Toutefois, les résultats relatifs aux instances SAT sont assez proches les uns des autres (97 pour PeneLoPe *freeze*, 95 pour ManySat, etc.), l'écart étant plus important pour les problèmes UNSAT.

En outre, nous avons également comparé ces solveurs sur une architecture composée de 32 cœurs. Plus précisément, la configuration matérielle est maintenant la suivante : Intel Xeon CPU X7550 (4 processeurs, 32 cœurs) 2.00GHz avec 18 Mo de mémoire cache et 256Go de mémoire vive. Chaque solveur est exécuté avec 32 *threads*, et les résultats obtenus sont présentés dans la Figure 4 de manière similaire.

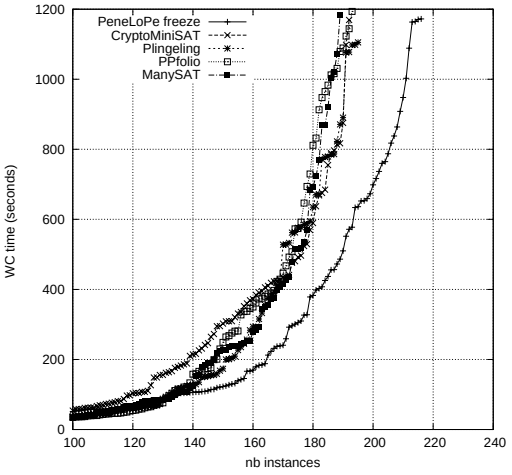
Tout d'abord, notons qu'à l'exception de plingeling, tous les solveurs améliorent leurs performances quand ils sont exécutés avec un nombre supérieur de *threads*. Le profit est cependant limité pour certains d'entre eux. Par exemple, cryptominisat résout 193 instances avec 8 *threads*, et 201 instances avec 32 *threads*. L'amélioration est bien supérieure avec PeneLoPe, dont les deux versions résolvent 15 instances supplémentaires, et plus spectaculaire encore pour ManySAT avec un gain de 29 instances. L'écart est tout particulièrement visible sur la Figure 4(c), puisque nos 3 compétiteurs résolvent le même nombre d'instances sur (environ) le même temps (les courbes de ppfolio, Plingeling et cryptominisat sont très proches les unes des autres), tandis que la courbe de PeneLoPe et celle de ManySAT montrent clairement leur capacité à résoudre un plus grand nombre de problèmes en un temps restreint. Il est d'ailleurs bon de noter que PeneLoPe résout le même nombre de problèmes que Plingeling, ppfolio et cryptominisat avec une limite (virtuelle) de temps de seulement 400 secondes. Enfin, il semble que le comportement de PeneLoPe puisse être amélioré sur les instances SAT. En effet, il apparaît que les redémarrages *luby* sont plus efficaces pour les problèmes possédant une solution que pour ceux qui en sont dépourvus, alors que le contraire se produit pour le redémarrage *lbd*.

Solver	#SAT	#UNSAT	#SAT+#UNSAT
`PeneLoPe` *freeze*	97	**119**	**216**
`PeneLoPe` *no freeze*	96	**119**	215
`plingeling` [4]	**99**	97	196
`ppfolio` [16]	91	103	194
`cryptominisat` [18]	89	104	193
`ManySat` [10]	95	92	187

(a) `PeneLoPe` VS l'état de l'art



(b) `PeneLoPe` *freeze* VS `Plingeling`



(c) Cactus plot

FIGURE 3 – Comparaison sur 8 cœurs

L'ajout d'unités de calcul, et par conséquent de fils d'exécution parallèles, a différents impacts. Par exemple, pour `ppfolio` et `plingeling`, le gain n'est pas énorme, puisque l'augmentation du nombre de *threads* ne fait qu'accroître le nombre de solveurs séquentiels explorant l'espace de recherche ; chaque *thread* ne profite pas ici du travail des autres, puisqu'au sein de ces solveurs, aucune collaboration (ou une collaboration très faible) n'est effectuée. `PeneLoPe` bénéficie mieux de l'augmentation de ressources, car le nombre de clauses échangées provenant de différents sous-espaces de recherche est supérieur. Ceci conduit à une connaissance plus grande pour chaque *thread*, sans ralentir le processus de recherche.

Pour finir, insistons sur le fait que durant nos expérimentation avec `PeneLoPe`, tous les *threads* possèdent *exactement* les mêmes paramètres et les mêmes stratégies, comme dans nos expérimentations préliminaires présentées dans la section 3. Offrir de la diversification aux différentes recherches CDCL concurrentes devrait accroître plus encore l'efficacité pratique de notre prototype.

6 Conclusion

Dans ce papier, nous avons proposé différentes stratégies permettant une meilleure gestion de l'échange de clauses apprises au sein de solveurs SAT parallèles de type *portfolio*. En se basant sur les concepts de *psm* et de *lbd* récemment proposés dans le cadre séquentiel, l'idée générale est d'adopter différentes stratégies pour l'importation et l'exportation de clauses. Nous avons étudié avec soin divers aspects empiriques des idées proposées, et comparé notre prototype aux meilleures implantations disponibles, montrant que notre solveur se révèle compétitif.

Assez clairement, diversifier le comportement exploratoire des *threads* devrait encore améliorer les performances de notre solveur, dans la mesure où cette diversification semble être la pierre angulaire de l'efficacité de certains solveurs comme `ppfolio`. Nous prévoyons d'étudier plus précisément cela dans un futur proche.

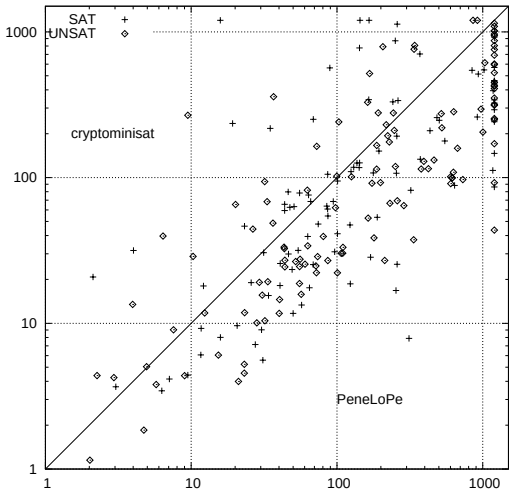
Références

[1] G. Audemard, J.M. Lagniez, B. Mazure, and L. Saïs. On freezeing and reactivating learnt clauses. Dans *proceedings of SAT*, pages 147–160, 2011.

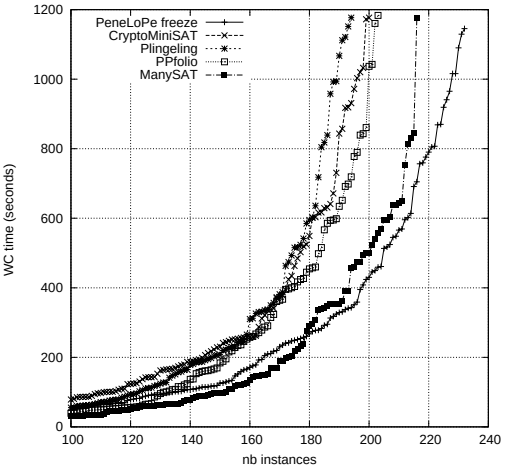
[2] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. Dans *proceedings of IJCAI*, pages 399–404, 2009.

Solver	#SAT	#UNSAT	#SAT+#UNSAT
PeneLoPe *freeze*	104	127	**231**
PeneLoPe *no freeze*	99	**131**	230
ManySAT [10]	105	111	216
ppfolio [16]	**107**	97	204
cryptominisat [18]	96	105	201
Plingeling [4]	100	95	195

(a) PeneLoPe VS l'état de l'art



(b) PeneLoPe *freeze* VS cryptominisat



(c) Cactus plot

FIGURE 4 – Comparaison sur 32 cœurs

[3] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. Dans *JAIR*, 22 :319–351, 2004.

[4] A. Biere. (p)lingeling. http://fmv.jku.at/lingeling.

[5] W. Chrabakh and R. Wolski. GrADSAT : A parallel SAT solver for the grid. Rapport technique, 2003.

[6] G. Chu, P. Stuckey, and A. Harwood. Pminisat : a parallelization of minisat 2.0. Rapport technique, SAT Race, 2008.

[7] A. Darwiche and K. Pipatsrisawat. *Complete Algorithms*, chapter 3, pages 99–130. IOS Press, 2009.

[8] Y. Hamadi, S. Jabbour, C. Piette, and L. Saïs. Deterministic parallel DPLL. Dans *JSAT*, 7(4) :127–132, 2011.

[9] Y. Hamadi, S. Jabbour, and L. Sais. Control-based clause sharing in parallel SAT solving. Dans *proceedings of IJCAI*, pages 499–504, 2009.

[10] Y. Hamadi, S. Jabbour, and L. Saïs. ManySat : a parallel SAT solver. Dans *JSAT*, 6 :245–262, 2009.

[11] A. Hyvärinen, T. Junttila, and I. Niemelä. Grid-based SAT solving with iterative partitioning and clause learning. Dans *proceedings of CP*, pages 385–399, 2011.

[12] S. Kottler and M. Kaufmann. SArTagnan - a parallel portfolio SAT solver with lockless physical clause sharing. Dans *Pragmatics of SAT*, 2011.

[13] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. Dans *proceedings of ISTCS*, pages 128–133, 1993.

[14] J. Marques-Silva and K. Sakallah. GRASP - A New Search Algorithm for Satisfiability. Dans *proceedings of ICCAD*, pages 220–227, 1996.

[15] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. Dans *proceedings of SAT*, pages 294–299, 2007.

[16] O. Roussel. ppfolio. http://www.cril.univ-artois.fr/~roussel/ppfolio.

[17] T. Schubert, M. Lewis, and B. Becker. Pamiraxt : Parallel sat solving with threads and message passing. Dans *JSAT*, 6(4) :203–222, 2009.

[18] Mate Soos. Cryptominisat. http://www.msoos.org/cryptominisat2/.

[19] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. Dans *proceedings of ICCAD*, pages 279–285, 2001.

Maintenance de valeurs alternatives dans les CSP dynamiques : principes et expérimentations en configuration de produit

Caroline Becker[1] **Hélène Fargier**[1*]

[1] IRIT, UMR 5505
Université Paul Sabatier, CNRS
118 Route de Narbonne, Toulouse

{becker, fargier}@irit.fr

Résumé

En configuration à base de contraintes, le catalogue est représenté par un CSP dont les solutions sont les différents produits configurés réalisables. Lors d'une session de configuration en ligne, le client cherche à définir le produit qui lui convient le mieux par affectation ou relaxation des différentes variables qui composent le produit configurable. À chaque étape d'interaction sont présentées, pour les variables non affectées, les valeurs de leurs domaines respectifs cohérentes avec les choix d'affectation déjà effectués sur les autres variables. La cohérence globale étant difficile à obtenir, ce filtrage est généralement établi par la maintenance d'un niveau de cohérence locale donné.

Dans ce travail, nous cherchons à rendre cette interaction plus conviviale en présentant aussi, pour les variables déjà affectées, les possibilités de changement de valeur d'affection, ou *valeurs alternatives* qui seraient cohérentes avec les autres choix effectués. Nous présentons donc ici une nouveau concept, celui de domaine alternatif d'une variable, et proposons une méthode pour calculer en une passe les domaines alternatifs de toutes les variables déjà affectées. De complexité au pire cas identique à la méthode "naïve" qui consisterait à simplement relaxer les choix portant sur la variable dont on veut calculer le domaine alternatif, cette méthode s'avère bien plus efficace expérimentalement.

Abstract

Constraint programming techniques are widely used to model and solve decision problems, and especially configuration problems. In this type of application, the configurable product is described by means of a set of constraint bearing on some configuration variable.

The user (typically, the customer) then interactively solves the CSP by assigning (and possibly, relaxing) the configuration variables according to her preferences. The aim of the system is then to keep the domains of the other variables consistent with these choices. Since the maintaining of global consistency is generally not tractable, the domains are generally filtered according to some level of local consistency.

In the present work, we aim at offering a more convenient level of interaction by providing the user with the alternative possible values of each of the already assigned variable - i.e. the values that could replace the current one without leading to a wipe out. We thus present the new concept of *alternative domains* in a (possibly) partially assigned CSP. We propose a propagation algorithm that computes all the alternative domains in a single step. Its worst case complexity is comparable with the one of the naive algorithm that would run a full propagation for each possible relaxation, but its experimental efficiency is much better.

*Ce travail est partiellement financé par l'ANR (projet BR4CP, ANR-11-BS02-008).

1 Introduction

La configuration de produits est une méthode de modélisation résultant de la personnalisation de masse (en anglais, *mass customization*) qui est un paradigme économique héritier de la production de masse (*mass production*) : nous sommes passés d'un article hautement standardisé, unique et produit en série à une gamme complexe et modulaire, c'est-à-dire une déclinaison du même produit [6]. Tout le problème alors,

pour le client, est de converger vers le produit le plus intéressant pour lui parmi l'ensemble, souvent hautement combinatoire, des produits réalisables.

Les approches par configuration à base de contraintes [9, 13, 7, 14, 15] proposent de représenter cette gamme par un problème de satisfaction de contraintes (CSP) sur un ensemble de variables de configuration et d'utiliser les algorithmes développés dans ce domaine, ou des variantes de ces algorithmes, pour fournir à l'utilisateur des fonctionnalités lui permettant de définir le produit configurable réalisable le plus en adéquation possible avec ses préférences.

Plusieurs travaux ont proposé d'utiliser des extensions des CSP pour formaliser des problèmes de configuration ; ces travaux traitent généralement de difficultés de représentation spécifiques à ces types de problèmes, comme par exemple le fait que l'existence d'une variable puisse dépendre de la valeur affectée à une autre [8]. L'étude des problèmes de configuration a ainsi poussé la communauté à étendre le modèle CSP de base, les CSP composites [12], les CSP interactifs [4], ou les CSP à hypothèses [1], etc.

Dans cet article, nous négligerons la prise en compte de ces difficultés de représentation pour nous tourner exclusivement vers l'interaction entre la machine et un client cherchant à définir un produit. Le principe de cette interaction est que l'utilisateur définit le produit qu'il désire en choisissant interactivement des valeurs pour les variables ; il peut également revenir en arrière en relâchant un ou plusieurs de ses choix. Après chaque action, les domaines de valeurs possibles fournis à l'utilisateur doivent être modifiés de manière à contenir toutes les valeurs compatibles avec les choix courants, et seulement les valeurs compatibles avec ces choix.

Nous proposons de rendre cette interaction plus pertinente en présentant non pas uniquement les domaines cohérents des variables non instanciées mais aussi les possibles changements de choix d'affectation, c'est-à-dire en présentant ce que nous appelons les domaines alternatifs des variables affectées.

Le présent article est structuré comme suit. La problématique des domaines alternatifs, et ses liens avec la notion de solution robuste proposée par Hebrard et al. [5] sont explicités dans la section 2. La section 3 présente notre approche algorithmique du calcul des domaines alternatifs. Nos premiers résultats expérimentaux sont donnés en section 4.

2 Problématique

Formellement, le produit configurable est donc représenté par un CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ et l'ensemble des choix courants par un ensemble de décision utilisateur, c'est-à-dire de couples (x_i, v) où x_i est une variable de $\mathcal{X}$ et $v \in D(x_i)$ est la valeur que l'utilisateur lui a affecté. Comme proposé par [1], le problème peut être représenté par un CSP à hypothèses :

Définition 1 (CSP à hypothèses) *Un CSP à hypothèses est un quadruplet $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ où :*

- *$(\mathcal{X}, \mathcal{D}, \mathcal{C})$ est un CSP ou $\mathcal{X} = \{x_1, \ldots, x_n\}$ est l'ensemble des variables, $\mathcal{D} = \prod_{j=1}^{N} D(x_j)$ le produit cartésien de leurs domaines initiaux et $\mathcal{C} = \{C_1, \ldots, C_m\}$ l'ensemble des contraintes. Une contrainte $C_i = (var(C_i), \mathcal{R}_i)$ est définie par l'ensemble $var(C_i) = \{x_{i_1}, \ldots, x_{i_k}\}$ des variables qu'elle impacte et par $\mathcal{R}_i \subset \prod_{j \in vars(C_i)} D(x_j)$ qui représente l'ensemble des valeurs que les variables contraintes peuvent prendre simultanément.*
- *$\mathcal{H}$ est un ensemble fini de contraintes unaires portant sur les variables de $\mathcal{X}$.*

Une décision utilisateur est un couple (x_i, v) où x_i étant une variable de $\mathcal{X}$ et $v \in D(x_i)$ la valeur que l'utilisateur lui a affecté. En configuration à base de contraintes, $\mathcal{H}$ représente l'ensemble des choix utilisateurs courants, avec une contrainte unaire par choix utilisateur ; les restrictions de $\mathcal{H}$ portent donc toutes sur des variables différentes. En pratique, on se limitera à des affectations et on notera $h_i = (x_i, v)$ la restriction de $\mathcal{H}$ portant sur x_i, si elle existe ; les principes et définitions proposées ici sont valides également quand $\mathcal{H}$ contient des restrictions quelconques de domaines.

Après chaque choix, le système filtre le domaine des variables, ne laissant idéalement dans les domaines que des valeurs compatibles avec les choix courants. En pratique, un niveau de cohérence plus faible est assuré, généralement l'arc cohérence, et on appelle *CSP courant* la fermeture, selon ce niveau de cohérence locale, du problème initial.

Définition 2 (Domaine courant)
Soit a un niveau de cohérence locale et $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ un CSP à hypothèses.
On appelle CSP courant *la fermeture par a-cohérence de $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H})$. On le notera $(\mathcal{X}, \mathcal{D}_c, \mathcal{C}_c)$.*
Le domaine courant *d'une variable x_i dans P est son domaine dans le CSP courant.*

Le domaine alternatif d'une variable est défini comme le domaine courant qu'aurait une variable instanciée si l'utilisateur remettait en cause cette affectation. C'est l'ensemble des valeurs que l'utilisateur peut donner à cette variable sans faire apparaître d'incohérence dans le CSP courant.

	1	2	3	4
x_1	$\star$	$\diamond$	$\diamond$	$\times$
x_2	$\times$	$\diamond$	$\diamond$	$\star$
x_3	$\times$	$\diamond$	$\diamond$	$\times$

TAB. 1 – Valeurs affectées ($\star$), interdites ($\times$) et alternatives ($\diamond$) pour le CSP $X = \{x_1, x_2, x_3\}, \mathcal{D} = D_1 \times D_2 \times D_3 = \{1, 2, 3, 4\}^3, \mathcal{C} = \{$ Alldiff$(x_1, x_2, x_3)\}, \mathcal{H} = \{(x_1, 0), (x_2, 4)\})$.

Définition 3 (Domaine alternatif)
Soit a un niveau de cohérence locale et $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ un CSP à hypothèses.
Le domaine alternatif d'une variable x_i modulo a, est son domaine selon la fermeture par a-cohérence du CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$.

Dans la suite, on utilisera la notation P_i^a pour désigner la fermeture par a-cohérence du CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$.

Une valeur v est donc une valeur alternative pour x_i soit si elle est dans le domaine courant de x_i (c'est en particulier le cas lorsque x_i est affectée à v), soit si x_i est affectée à une autre valeur que v et que le seul retrait de cette affectation suffit à réautoriser v. Par exemple, si x_i est la dernière variable à avoir été affectée, toutes les valeurs de son domaine avant l'affectation sont dans son domaine alternatif. Un exemple de domaine alternatif peut être visualisé sur la figure 1.

La notion de domaine alternatif peut être rapprochée de la notion de $(1, 0)$ super-solution proposée par Hebrard et al. [5]. Rappelons qu'une (i, j) super solution d'un CSP est une solution complète d du CSP, c'est-à-dire une affectation de toutes les variables, telle que quel que soit l'ensemble $Y \subseteq X$ de i variables que l'on considère, il existe une solution d' du CSP telle que, (i) les valeurs données aux variables de Y diffèrent dans d et d' et (ii) d et d' diffèrent sur au plus $i + j$ variables au total. Cette notion généralise la notion de *fault tolerant solution* proposée par [16] - une *fault tolerant solution* étant une $(1, 0)$ super solution, c'est-à-dire une solution du CSP pour laquelle chaque variable affectée possède une valeur alternative : si le choix utilisateur devient impossible pour une raison ou pour une autre, on peut le "réparer" sans changer les valeurs des autres variables. En d'autres termes, une fois l'affectation complète d réalisée, chaque variable x_i doit posséder un domaine alternatif contenant au moins une valeur en plus de $d[x_i]$. Plus formellement, on peut montrer que :

Propriété 1
Soit d une affectation de toutes les variables d'un $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ et $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ le CSP à hypothèses où $\mathcal{H} = \{h_i = (x_i, d[x_i]), i = 1, card(\mathcal{X})\}$.
d est $(1, 0)$-super solution de $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ si et seulement si pour tout $x_i \in \mathcal{X}$, le domaine alternatif de x_i par arc cohérence ce est un sur ensemble strict de $\{d[i]\}$

Preuve : Soit d une affectation complète de X ; on note $d_{i \leftarrow v}$ l'affectation d' telle que $d[x_i] = v$ et que, pour tout $j \neq i$, $d'[x_j] = d[x_j]$. Considérons le CSP à hypothèses $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ correspondant : puisque toutes les variables sont affectée, $\forall i = 1, card(\mathcal{X}), h_i = (x_i \leftarrow d[x_i])$.

Supposons que pour tout x_i le domaine alternatif de x_i par arc cohérence est un sur ensemble strict de $\{d[i]\}$, i.e. que $D_{alt}^{arc}(x_i) \supsetneq \{d[x_i]\}$, et considérons une variable x_i quelconque ; par définition de la notion de domaine alternatif, le domaine de x_i dans P_i^{arc}, la fermeture par arc cohérence de $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H} \setminus \{h_i\})$ contient au moins une valeur v différente de $d[x_i]$. Puisque $\mathcal{H}$ affecte toutes les variables, chacune des variables $x_j \neq x_i$ de P_i^{arc} est affectée à la valeur $d[x_j]$. Par définition de l'arc cohérence, et puisque que seule x_i est non affectée le domaine alternatif de x_i contient donc toutes les valeurs globalement cohérentes avec cette affectation ; $d_{i \leftarrow v}$ est donc une solution de $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. Ce raisonnement s'appliquant à chaque, x_i, d est donc une $(1, 0)$ super solution de $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

Réciproquement, supposons que d est une $(1, 0)$ super solution de $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. Par définition de cette notion, pour chaque x_i, il existe une valeur $v \neq d[x_i]$ telle que $d_{i \leftarrow v}$ soit une solution du CSP. Comme l'arc cohérence ne filtre que des valeurs ne conduisant à aucune solution, v appartient au domaine de x_i dans P_i^{arc} : le domaine alternatif de x_i contient donc, en plus de $d[x_i]$, la valeur v : c'est donc sur ensemble strict de $\{d[i]\}$.

□

La différence fondamentale entre les notions de $(1, 0)$ super-solution et de domaine alternatif est que la première notion est limitée aux affectations complètes, alors que la seconde propose des valeurs de réparation y compris pour des configurations partielles. Les algorithmes de calcul de super-solutions fondés sur la duplication de variable ne fonctionnent pas pour la maintenance de domaines alternatifs, comme le montre l'exemple de la figure 2 : lorsque l'utilisateur a choisi $x_1 \leftarrow 1$ et $x_2 \leftarrow 2$, la propagation réduit le domaine de x_3 à $\{3\}$; et celui de x_1', qui devrait porter les valeurs alternatives de x_1, est alors vidé, alors que 3 fait bien parti du domaine alternatif de x_1.

Notons enfin que l'objectif pratique est différent : dans le premier cas, on cherche quelques solutions robustes (mais pas toutes : elles sont potentiellement

en nombre exponentiel), et seulement des solutions robustes alors que dans le second, on désire calculer *toutes* les valeurs alternatives, si il y a.

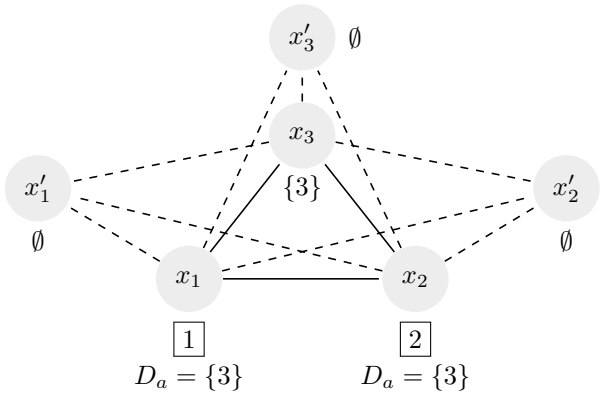


FIG. 1 – Représentation de $x_1 \neq x_2$, $x_2 \neq x_3$, $x_1 \neq x_3, \forall i, D(x_i) = \{1, 2, 3\}$ lorsque l'utilisateur a choisi $x_1 \leftarrow 1$ et $x_2 \leftarrow 2$.

Le domaine alternatif est une notion orthogonale à la notion d'explications de retraits, telles que celle proposée par PaLM [11]. En effet, les explications servent à justifier de la restriction de domaines et à proposer une stratégie de restauration de ces valeurs par relâchement des choix déjà effectués, alors que la présentation du domaine alternatif permet justement d'effectuer des restaurations de valeur sans avoir besoin de modifier les autres variables. Disons qu'une valeur est alternative si son retrait n'a d'autre explication que son l'affectation de sa variable : la notion est moins lourde, et son calcul ne nécessite pas la mémorisation d'explications.

3 Calcul des domaines alternatifs

Lorsque n variables sont affectées, une méthode naïve pour calculer leurs domaines alternatifs respectifs consiste en la génération de $n+1$ copies du CSP : un CSP de référence P_0 où toutes les variables sont instanciées, et n CSP P_i, chaque P_i ayant les mêmes affectations que P_0, sauf pour la variable x_i. Chacun des P_i est alors filtré par a-cohérence. Cette méthode, qui a l'avantage de demander peu d'espace mémoire supplémentaire, servira de point de référence par rapport auquel comparer l'algorithme que nous présentons, qui prend le parti exactement inverse, à savoir stocker les informations utiles afin d'éviter des calculs redondants.

Si, lors de la présentation de la notion de domaine alternatif, on ne faisait aucune supposition sur les domaines des variables qui pouvaient être continus ou discrets, nous supposons par la suite que le domaine de chaque variable est fini.

3.1 Retraits et justification suffisantes

L'idée est de maintenir, pour chaque retrait de valeur d'un domaine, une liste de flags : un par affectation $h_i \in \mathcal{H}$. Ce flag est positionné à vrai si et seulement si la remise en cause de l'affectation provoque le retour de la valeur dans le domaine de sa variable.

Définition 4 (Retrait)
Soit $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ un CSP à hypothèses et $(\mathcal{X}, \mathcal{D}_c, \mathcal{C}_c)$ la fermeture par a−cohérence de $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H})$.
Un retrait est un couple (x_j, v) tel que $v \in D(x_j)$ et $v \notin D_c(x_j)$
On note $\mathcal{R}$ est l'ensemble des retraits de P.

Pour plus de clarté, un retrait (x_j, v) sera souvent noté $(x_j \neq v)$

Définition 5 (Justification Suffisante)
Soit $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ un CSP à hypothèses.
Pour toute valeur v dans le domaine initial d'une variable x_j, et tout $h_i \in \mathcal{H}$, on dit que h_i est une justification suffisante de v pour x_j modulo une propriété de cohérence locale *a (ou "justification a-suffisante") si v appartient au domaine de x_j dans la fermeture par a-cohérence de $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$.*
On dit que h_i est une justification suffisante d'un retrait $(x_j \neq v) \in \mathcal{R}$ ssi c'est une justification a-suffisante de v pour x_j.

Typiquement, si la dernière affectation a provoqué l'élimination de v pour x_j, elle est une justification suffisante de $(x_j \neq v)$.

De la même façon, les retraits invalident des tuples qui sont eux-mêmes supports pour des valeurs, et la notion de justification suffisante peut être étendue aux tuples. Formellement :

Définition 6 (Ensemble de conflits, tuple valide)
Un retrait $(x \neq v)$ et un tuple t sont en contradiction directe ssi $t[x] = v$.
L'ensemble des conflits d'un tuple est l'ensemble des retraits avec lesquels il est en contradiction directe : $CS(t) = \{(x_i \neq v) \in \mathcal{R}$ t. q. $t[x_i] = v\}$.
Un tuple est valide ssi il n'est en contradiction directe avec aucun retrait (dans les cas contraire, il est dit invalide).

Définition 7 (Justification suffisante d'un tuple)
Soit C in contrainte de $\mathcal{C}$ et $t \in R_C$ un tuple satisfaisant C.
On dit que h_i est une justification a-suffisante pour

t ssi, pour tout $x_j \in vars(C)$, $t[j]$ appartient au domaine de x_j dans la fermeture par a-cohérence de $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$.

Au sens de ces définitions, h_i est également une justification suffisante de t (respectivement de v pour x_j) si t est valide (resp. si v est dans le domaine courant x_j). En fait, pour les tuples valides (et les valeurs des domaines courants), toute hypothèse est une justification suffisante.

Notre algorithme est fondé sur le fait que l'affectation h_i est une justification a-suffisante pour le tuple t ssi, pour chaque variable x_j concernée, soit $t[x_j]$ est dans le domaine courant de x_j, soit h_i est une justification suffisante de chacune des valeurs du tuple qui ne sont pas dans les domaines courants. C'est le sens de la propriété suivante :

Propriété 2 *h_i est une justification suffisante d'un tuple invalide t ssi c'est une justification suffisante de tous les retraits de son ensemble de conflits.*

Preuve :
$\boxed{\Rightarrow}$ Supposons par l'absurde qu'il existe h_i justification a-suffisante d'un tuple t et $(x \neq v)$ un retrait appartenant à l'ensemble de conflits de t (i.e. $t[x] = v$) dont h_i ne soit pas une justification suffisante.
On note P_i^a la fermeture par a-cohérence de $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$. Puisque h_i est une justification a-suffisante de t , par définition, t est valide P_i^a. D'un autre coté, puisque h_i n'est pas une justification suffisante de $(x \neq v)$, v n'est pas dans le domaine de x dans P_i^a ; t est donc invalide dans P_i^a, de qui contredit la déduction précédente.
$\boxed{\Leftarrow}$ Réciproquement, soit h_i une justification a-suffisante de tous les retraits de l'ensemble des conflits d'un tuple t. Pour chacun de ces $(x_j \neq v_j)$, v_j est par définition dans le domaine de x_j dans P_i^a. Donc t est un tuple valide dans P_i^a - par définition de la notion, h_i est donc une justification suffisante de t.
□

On peut montrer que, lorsque la propriété de cohérence locale à assurer est l'arc-cohérence généralisée :

Propriété 3 *h_i est une justification suffisante d'un retrait $(x \neq v)$ modulo l'arc-cohérence (GAC) ssi, pour chacune des contraintes qui portent sur x, il existe un tuple t support de $x = v$ dont h_i soit une justification GAC-suffisante.*

Preuve :

$\boxed{\Rightarrow}$ Soit h_i est une justification suffisante d'un retrait $(x \neq v)$ modulo l'arc-cohérence (GAC). Supposons par l'absurde une contrainte C portant sur la variable x telle que h_i ne soit une justification suffisante d'aucun des tuples $t \in R_C$ supports de $x = v$, ce qui signifie que tous ces t sont invalides dans P_i^{GAC}. Donc v n'a aucun support par arc cohérence sur C dans P_i^{GAC} : il n'est donc pas dans le domaine de x dans la fermeture par arc cohérence de P_i^{GAC} : h_i n'est pas une justification suffisante de $(x \neq v)$.

$\boxed{\Leftarrow}$ Réciproquement, considérons une hypothèse h_i et supposons que, $\forall C$ portant sur x, $\exists t$ support de $x = v$ dont h_i soit une justification GAC-suffisante, c'est à dire que pour chacune contrainte C portant sur x, il existe une support t de $x = v$ valide dans P_i^{GAC} ; par définition de l'arc cohérence, v appartient donc au domaine de x dans P_i^{GAC}, i.e.
h_i est une justification GAC-suffisante de r. □

Des propriétés similaires peuvent être établies pour d'autres niveaux de cohérence locale induisant des retraits dans les domaines :

Définition 8
Soit v une valeur dans le domaine initial d'une variable x, $\mathcal{V}$ un ensemble de k variables et $\mathcal{C}_k$ l'ensemble des contraintes qui portent sur $\{x\} \cup \mathcal{V}$.
v est localement cohérente par rapport à $\mathcal{V}$ ssi il existe une affectation t de $\{x\} \cup \mathcal{V}$ telle que $t[x] = v$ et $\forall C \in \mathcal{C}_k, t[vars(C)] \in R_C$, où $t[vars(C)]$ est la projection de t sur les variables de C.

On dit que t est un support de v sur $\mathcal{V}$.

Définition 9
Un CSP est $(1,k)$ cohérent ssi, pour toute variable et toute valeur dans son domaine, pour tout groupe $\mathcal{V}$ de k variables, v admet un support sur $\mathcal{V}$.

Propriété 4
h_i est une justification $(1,k)$-suffisante d'un retrait $(x \neq v)$ ssi, pour chaque groupe $\mathcal{V}$ de k variables il existe support t de v dont h_i est justification $(1,k)$-suffisante.

Preuve : $\forall (x_1, ..., x_k), \exists (v_1, ..., v_k)$ support de $x = v$ tel que h_i justification $(1,k)$-suffisante de $(v_1, ..., v_k)$
$\Leftrightarrow \forall (x_1, ..., x_k), \exists (v_1, ..., v_k)$ support de $x = v$ tel que chacun des v_j appartienne au domaine de x_j dans la fermeture par $(1,k)$-cohérence de $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$
$\Leftrightarrow$ v appartient au domaine de x dans la fermeture par $(1,k)$-cohérence de $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$ (par définition de la $1,k$ cohérence)

$\Leftrightarrow h_i$ est une justification $(1,k)$-suffisante de $x \neq v$
□

3.2 Algorithme de maintenance des domaines alternatifs

		supports de $x = v$			
$\mathcal{R}$	justifications	t_1	t_2	t_3	t_4
$x_1 \neq v_1$	h_1, h_2	$\star$	$\star$		
$x_2 \neq v_2$	h_1, h_3	$\star$			$\star$
$x_3 \neq v_3$	h_2, h_4		$\star$	$\star$	$\star$
résultante	h_1, h_2, h_4	h_1	h_2	h_2, h_4	$\emptyset$

TAB. 2 – Calcul du tableau de flags d'un retrait ($x \neq v$) sur une contrainte C ; les t_i sont les supports de $x = v$. Un $\star$ dans la case $(x_j \neq v_j, t_i)$ signifie que le tuple t_i est invalide lorsque $x_j \neq v_j$.

Notre algorithme est détaillé ci dessous (Algorithmes 1 et 2). C'est un algorithme de type GAC4 [10] - la différence est qu'une valeur retirée va éventuellement être retirée plusieurs fois (à cause de plusieurs contraintes), et que cette nouvelle cause de retrait doit être propagée. Plus largement, un retrait ($x \neq v$) doit être propagé à chaque modification de son vecteur de justifications. I. La modification de ce vecteur, comme celle des vecteurs de flag des tuples, étant monotone (une justification supposée suffisante peut disparaître, mais jamais redevenir suffisante), l'algorithme termine. Sans le mémoriser forcément, on peut parler des flags d'un tuple : c'est un vecteur de bits, qui, pour chaque h_k, porte la valeur vrai ssi h_k est une justification suffisante pour chacune des valeurs du conflit set de t.

En effet, pour que le choix utilisateur h_k soit une justification suffisante de $x \neq v$ sur C, il faut que la remise en cause de h_k aboutisse à ce qu'au moins un tuple de C compatible avec $x = v$ devienne à nouveau valide ; et un tuple t redevient valide par la remise en cause de h_k ssi tous les éléments de son ensemble de conflit ont h_k comme justification suffisante. En d'autre termes les flags d'un tuple est l'intersection des flags des retrait qui contredisent le tuple. Finalement lorsque c'est une propriété d'arc cohérence qui est maintenue, on obtient la propriété 5 :

Propriété 5

$$f_{(x \neq v)} = \bigwedge_{C, x = out(C)} \bigvee_{t \in Support(x,v,C)} \bigwedge_{r' \in CS(t)} f(r')$$

Preuve : L'ensemble des justifications AC-suffisantes d'un retrait est, d'après la proposition 3,

$$f_{(x \neq v)} = \bigwedge_{C, x = out(C)} \bigvee_{t \in Support(x,v,C)} f(t)$$

Or, d'après la proposition 2, l'ensemble des justifications AC-suffisantes d'un tuple est l'intersection des justifications AC-suffisantes des retraits de son ensemble de conflits. □

L'algorithme 2, inspiré de l'algorithme GAC4 [10] décrit la mise à jour des vecteurs de flags des retraits selon cette formule. Nous notons :

- $Cpt(x, v, C)$, le nombre de tuples supports de $x = v$ pour la contrainte C .
- $f_{(x_k \neq u)}$, l' ensemble de justifications suffisantes du retrait $(x_k \neq u)$. $f_{(x_k \neq u)}[h_k] = true \Leftrightarrow h_k$ est une justification suffisante de $x_k \neq u$. Vaut initialement Vrai^n
- $D_c(x_i)$ le domaine courant de la variable x_i.
- $D(x_i)$ le domaine initial de la variable x_i.
- $S_{i,v,C}$, l'ensemble des tuples t autorisés par la contrainte C tels que $t[i] = v$.
- $f(t)$: Ensemble des justifications suffisantes du tuple t. Initialement, vaut Vrai^n
- $valide(t)$: mémorise si le tuple t est valide dans le CSP courant.

Procedure Initialise($(\mathcal{X}, \mathcal{D}_c, \mathcal{C}_c)$: CSP initial)
/* $(\mathcal{X}, \mathcal{D}_c, \mathcal{C}_c)$ supposé arc cohérent */
/* Tous les tuples sont supposés valides */
begin
  pour tous les *Contraintes* C **faire**
    pour tous les $x_i \in vars(C)$, $v \in D(x_i)$ **faire**
      $Cpt(x_i, v, C) := 0$;
    fin
    pour tous les $t \in vars(C)$ **faire**
      $f_t = \text{Vrai}^n$;
      $valide(t) = \text{Vrai}$;
      $CS(t) = \text{Faux}^n$;
      pour tous les $x_i \in vars(C)$ **faire**
        $Cpt(x_i, t[i], C) + +$;
      fin
    fin
  fin
end

Algorithm 1: Initialisation

Au lieu de rentrer une seule fois dans la boucle Q, comme dans un GAC4 classique, chaque retrait peut rentrer n fois, autant qu'il y a de changements possible dans un vecteur de flags. Le surcoût des calculs de vecteurs est négligeable (on les effectue en temps constant par conjonction ou disjonction de bitvecteurs), la complexité en pire cas de notre algorithme $O(n.e.d^k)$ où e est le nombre de contraintes, m le nombre de variables, n le nombre d'hypothèses , d la taille maximale des domaines et k l'arité maximale d'une contrainte. C'est

```
Procedure Propagate( (x_k, w) : hypothèse ;
(X, D_c, C_c) : CSP courant) Q := ∅;
pour tous les u ≠ w ∈ D_c(x_k) faire
    f_(x_k≠u) ← Faux^n;
    si u ∈ D_c(x_k) alors
        f_(x_k≠u)[h_k] ← Vrai;
    fin
    Ajouter ((x_k ≠ u)) à Q;
fin
pour Q non vide faire
    Choisir et retirer un nœud (x_i ≠ v) de Q;
    si v ∈ D_c(x_i) alors
        Retirer v de D_c(x_i);
    fin
    pour tous les C portant sur x_i et tout tuple
    t de S_i,v,C faire
        Mem ← f_t;
        f_t ← f_t ∧ f_(x_i≠v);
        si valide(t) alors
            pour tous les x_j ≠ x_i ∈ vars(t) faire
                Cpt(x_j, t[j], C)^−−;
                si Cpt(x_j, t[j], C) = 0 alors
                    f_(x_i≠v) ← Vrai^n  /* init */ ;
                    Ajouter ((x_j ≠ v'), C) à Q;
                fin
            fin
            valide(t) = false;
        fin
        si Mem! = f_t  Une justif de t a disparu alors
            pour tous les x_j ≠ x_i ∈ vars(t) faire
                mem' = f_x_j≠t[j];
                f_x_j≠t[j],C = f_x_j≠t[j],C ∨ f(t);
                f_x_j≠t[j] = f_x_j≠t[j] ∧ f_x_j≠t[j],C;
                si mem' ≠ f_x_j≠t[j] alors
                    Ajouter (x_j ≠ t[j]) à Q;
                fin
            fin
        fin
    fin
fin
```

Algorithm 2: Propagation de la décision utilisateur $h_k = (x_k \leftarrow w)$

donc la même que celle de la méthode naïve fondée sur GAC-4, dont notre algorithme est inspiré : $n.O(e.d^k)$. A ceci près que, dans la méthode naïve, GAC-4 tourne exactement n fois, alors qu'ici il tourne au plus n fois.

En termes d'espace, sont mémorisés par GAC4 les ensembles $S_{i,v,C}$ de supports de chaque valeur - une structure qui est en $O(A)$, A étant l'espace pris pas le tuples admissibles contenus dans les tables des contraintes. Notre algorithme maintient de plus, pour chaque tuple t, un vecteur de n booléennes (les flags) soit un espace $O(T.n)$, T étant le nombre de tuples admissibles . Le problème produit au plus autant de retraits que de valeurs dans les domaines, pour chacun desquels on mémorise également un vecteur de n booléens - donc une structure en $O(d.m.n)$. Soit par rapport à l'algorithme de base, GAC4, un espace supplémentaire bornée par $O(n.(T + d.m))$.

4 Résultats Expérimentaux

Nous avons testé ces algorithmes sur un benchmark issus du monde industriel. Il relativement petit (32 variables, de domaines de taille 2 à 10, et 35 contraintes binaires)[1]. Il s'agit d'une souffleuse, c'est-à-dire une machine créant de manière automatique des bouteilles dans différentes matériaux.

Chaque point de l'échantillon simule une session de configuration. Il consiste en une série d'affectations, selon un ordre aléatoire, de toutes les variables du problème. A chaque affectation de variable, les domaines des variables libres sont filtrés par arc cohérence et les domaines alternatifs de toutes les variables déjà affectées sont calculés. L'ordre dans lequel les variables sont affectées est choisi selon un distribution uniforme, et pour chaque variable, la valeur affectée est choisie de manière équiprobable parmi les valeurs restant dans son domaine (i.e. choisie dans son domaine courant). Notre échantillon contient 1000 séquences d'affectation.

Pour chaque séquence $(x_1, \ldots, x_m\}$ nous mesurons, à chaque affectation $x_k = v_k$, le temps nécessaire pour calculer les domaines alternatifs des variables précédemment affectées (i.e. x_1 à x_i). Tout le processus est répété pour les deux algorithmes, qui jouent donc sur les mêmes séries d'affectations.

La figure 2 présente les résultats de cette expérimentation. On a porté en abscisse la position de l'affectation dans la séquence. La courbe en plein représente le temps moyen nécessité par la méthode naïve (croissante, ronds), celle en pointillé (stable, carrés) la moyenne des temps nécessités de notre algorithme.

[1] Il peut être retrouvé sur `ftp://ftp.irit.fr/pub/IRIT/ADRIA/PapersFargier/Config/___Souffleuse.csp`

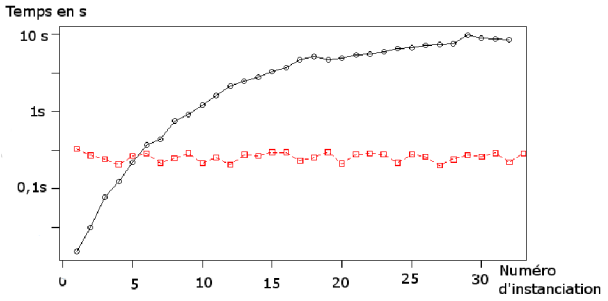


FIG. 2 – Comparaison des temps de calcul de la méthode naïve (en noir, ∘, croissante) et de notre algorithme (en rouge □, stable.).

Les résultats sont excellents : l'algorithme que nous proposons offre des résultats plus rapides dès que plus de 5 variables sont affectés - i.e. dès que le nombre de domaines alternatifs à calculer dépasse 5. Comme on pouvait s'en douter, le temps nécessité par l'algorithme naïf croît avec le nombre de variables concernées, alors que le temps nécessité par notre algorithme reste stable.

5 Conclusion et perspectives

Dans ce travail, nous avons proposé un nouveau concept, celui de domaine alternatif d'une variable selon un niveau de cohérence locale. Nous avons présenté, pour la maintenance des domaines alternatifs modulo l'arc cohérence généralisée, une extension de l'algorithme GAC4 qui consiste en une prolongation de la propagation des retraits au delà des domaines des variables concernées. Par opposition à une méthode naïve qui consisterait à effectuer from scratch autant de propagations qu'il y a de domaines alternatifs à calculer, notre méthode mémorise des justifications (limitées) des retraits. Elle consomme plus d'espace mémoire que la méthode naïve et nécessite des contraintes définies en extension. Nos premières expérimentations, sur un cas réel, suggèrent qu'elle est rapidement plus efficace que la méthode naïve.

L'espace nécessaire reste une limite de la méthode ; le surcoût en espace, par rapport à l'algorithme GAC4 dépend directement du nombre de variables dont on veut calculer le domaine alternatif. Cela dit, il faut garder à l'esprit que l'on ne demandera pas en pratique au système de présenter les domaines alternatifs de toutes les variables : l'utilisateur est opérateur humain avec une charge mentale limitée et il n'est pas évident qu'il puisse (ou même désire) se représenter efficacement un grand nombre de domaines alternatifs. Dans une application de configuration par exemple, on ne présentera les domaines alternatifs que de certaines variables, par exemple celles correspondant au sous composant que l'on est en train de configurer.

Les concepts développés dans cet article, nous l'avons vu, sont assez proches des notions de solution robuste et de restauration de valeur. Dans le travail présent, nous nous sommes focalisés sur le calcul des domaines alternatifs. Cela dit, une partie des information utilisées par l'algorithme peut mieux être exploitée lors de l'interaction avec l'utilisateur : une justification suffisante d'un retrait ($x \neq v$) peut être vue comme une restauration de taille 1 de v, et être présentée en tant que telle lorsque l'utilisateur cherche à libérer une valeur interdite par les choix courants.

Ce travail suggère de nombreux développements et perspectives. Tout d'abord, il faut évidement améliorer notre algorithme, par exemple en proposant une version paresseuse, et compléter nos expérimentations. Il faudra ensuite réfléchir à ce qui peut être fait pour le maintien des domaine alternatifs dans les CSP qui ne limitent pas leurs contraintes à des représentations sous forme de table. Enfin et surtout, il faudrait savoir prendre en compte l'interaction dans son ensemble : pour l'instant, nous n'avons considéré que l'affectation de valeur ; reste à traiter la relaxation par l'utilisateur d'un ou plusieurs de ses choix. Cette adaptation supposera une hybridation avec les algorithmes de maintient de cohérence dans les CSP dynamiques [2, 3].

Références

[1] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic csps application to configuration. *Artificial Intelligence*, 135(1-2) :199–234, 2002.

[2] Christian Bessière. Arc-consistency in dynamic constraint satisfaction problems. In *Proceedings of AAAI'91*, pages 221–226, 1991.

[3] Romuald Debruyne. Arc-consistency in dynamic csps is no more prohibitive. In *Proceedings ICTAI'96*, pages 299–307, 1996.

[4] Ester Gelle and Rainer Weigel. Interactive configuration using constraint satisfaction techniques. In *Proceedings of PACT-96*, pages 37–44, 1996.

[5] Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. Super solutions in constraint programming. In *Proceedings of CPAIO'04*, pages 157–172, 2004.

[6] Hanna Kropsu-Vehkapera, Harri Haapasalo, Olli Jaaskelaine, and Kongkiti Phusavat. Product configuration management in ict companies : The

practitioners' perspective. *Technology and Investment*, 2(4) :0–0, 2011.

[7] Daniel Mailharro. A classification and constraint-based framework for configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 12 :383–397, September 1998.

[8] Sanjay Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of AAAI'90*, pages 25–32, 1990.

[9] Sanjay Mittal and Felix Frayman. Towards a generic model of configuraton tasks. In *Proceedings of the IJCAI'89*, pages 1395–1401, 1989.

[10] Roger Mohr and Gérald Masini. Good old discrete relaxation. In *Proceedings of the ECAI'88*, pages 651–656, 1988.

[11] Samir Ouis, Narendra Jussien, and Olivier Lhomme. Explications conviviales pour la programmation par contraintes. In *Actes de JFPLC*, pages 105–118, 2002.

[12] Daniel Sabin and Eugene C. Freuder. Configuration as composite constraint satisfaction. In *AI and Manufacturing Research Planning Workshop*, pages 153–161, 1996.

[13] Daniel Sabin and Rainer Weigel. Product configuration frameworks — a survey. *IEEE Intelligent Systems*, 13(4) :42–49, 1998.

[14] Markus Stumptner, Gerhard E. Friedrich, and Alois Haselböck. Generative constraint-based configuration of large technical systems. *AI EDAM*, 12(04) :307–320, 1998.

[15] Junker Ulrich. Configuration. In *Handbook of Constraint Programming*, pages 837–874. Elsevier Science, 2006.

[16] Rainer Weigel and Christian Bliek. On reformulation of constraint satisfaction problems. In *Proceedings of ECAI'98*, pages 254–258, 1998.

Une nouvelle sémantique pour la programmation logique capturant la sémantique des modèles stables : la sémantique des extensions

Belaïd Benhamou et Pierre Siegel

Université d'Aix-Marseille
Centre de Mathématiques et d'Informatique
39, rue Joliot Curie - 13453 Marseille cedex 13, France
{Belaid.Benhamou;siegel}@cmi.univ-mrs.fr

Abstract

Answer set programming is a well studied framework in logic programming. Many research works had been done in order to define a semantics for logic programs. Most of these semantics are iterated fixed point semantics. The main idea is the canonical model approach which is a declarative semantics for logic programs that can be defined by selecting for each program one of its canonical models. The notion of canonical models of a logic program is what it is called the stable models. The stable models of a logic program are in a certain sense the minimal Herbrand models of its "reduct" programs. Here we introduce a new semantics for logic programs that is different from the known fixed point semantics. In our approach, logic programs are expressed as CNF formulas (sets of clauses) of a propositional logic for which we define a notion of extension. We prove in this semantics, that each consistent CNF formula admits at least an extension and for each given stable model of a logic program there exists an extension of its corresponding CNF formula which logically entails it. On the other hand, we show that some of the extensions do not entail any stable model, in this case, we define a simple descrimination condition which allows to recognize such extensions. These extensions could be very important, but are not captured by the stable models semantics. Our approach, extends the stable model semantics in this sense. Following the new semantics, we give a full characterization of the stable models of a logic program by means of the extensions of its CNF encoding verifying a simple condition, and provide a procedure which can be used to compute such extensions from which we deduce the stable models of the given logic program.

Résumé

La programmation par ensembles réponses (Answer Set Programming) est un cadre bien étudié en programmation logique. Plusieurs travaux ont été faits pour définir une sémantique pour les programmes logiques. La plupart de ces sémantiques sont en fait des sémantiques de point fixe. L'idée principale est le calcul de modèles canoniques du programme logique considéré, appelés modèles stables. Les modèles stables sont dans un certain sens des modèles minimaux des programmes réduits. Nous introduisons une nouvelle sémantique pour les programmes logiques, à partir d'une notion d'extension d'une formule propositionnelle classique. Ces extensions peuvent être calculés de manière itérative. Un programme logique est alors codé par un ensemble de clauses de la logique propositionnelle. On prouve que chaque formule consistante admet au moins une extension et que, pour chaque modèle stable d'un programme logique, il existe une extension de son codage qui l'implique logiquement. Certaines des extensions ne correspondent pas à un modèle stable mais sont intéressantes. Nous donnons une condition discriminante simple qui permet de reconnaitre de telles extensions. Enfin, nous décrivons un algorithme qui calcule les extensions de la formule CNF codant le programme logique. De cet ensemble d'extension on peut extraire les modèles stables du programme logique initial.

1 Introduction

L'étude proposée ici se situe dans le cadre de la programmation par ensemble réponse (ASP). Ce cadre peut être vu comme un cas particulier de la logique des défauts [12]. Une des questions importantes de l'ASP est de définir une sémantique pour les programme logiques.

Un programme logique π est un ensemble de rè-

gles de la forme $r : concl(r) \leftarrow prem(r)$, où $prem(r)$ est l'ensemble des prémisses de la règle. Cet ensemble de prémisses est une conjonction de littéraux qui peut contenir des négations classiques et des négations par échec. La partie gauche $concl(r)$ est la conclusion de la règle, exprimée en général par un seul littéral positif, ou dans certains cas par une disjonction de littéraux positifs (programmes logiques disjonctifs). L'ensemble $prem(r)$ peut être aussi appelé corps de la règle r et $concl(r)$ appelé tête de la règle ($r : head(r) \leftarrow body(r)$). Chaque programme logique π est traduit en un programme logique terminal équivalent $Ground(\pi)$ par le remplacement de chaque règle contenant des variables par toutes ses instances terminales, de sorte que chaque littéral de $Ground(\pi)$ est terminal. Cette technique est utilisée pour éliminer les variables même si le programme contient des symboles fonctionnels et si son univers de Herbrand est infini.

Parmi les sémantiques les plus connues pour les programmes logiques, on a la complétion de Clark [1], et la sémantique des modèles stables (ensembles de réponses) [7]. On sait que chaque modèle stable d'un programme logique est un modèle de sa complétion, mais que l'inverse n'est pas toujours vrai. Fages [6] a montré que les deux sémantiques sont équivalentes pour les programmes logiques sans boucle (tight programs). Une généralisation du résultat de Fage aux programmes logiques avec d'éventuelles expressions imbriquées de négations par échec dans les corps de leurs règles a été donné dans [5]. D'autre part, Fangzhen Lin et Zhao Yutin ont proposé dans [3] d'ajouter à la complétion d'un programme logique, des formules appelées *loop formulas*. Ils ont montré que l'ensemble des modèles de la complétion étendue par ces dernières formules est identique à l'ensemble des modèles stables du programme logique considéré même si ce dernier contient des boucles (not tight).

La quasi-totalité des sémantiques connues pour les programmes logiques sont déclaratives et/ou sont des sémantiques de point fixe (comme pour la logique des défauts un modèle stable est un ensemble de littéraux qui vérifie une équation qui peut ne pas avoir de solution). Ici nous introduisons une nouvelle sémantique qui n'est ni déclarative ni définie par un point fixe classique (on a toujours une solution). Cette sémantique peut être considérée comme un cas particulier de la logique des hypothèses [14, 13] qui, en particulier, étend la logique des défauts. La sémantique est basée sur une notion d'extension d'une formule propositionnelle classique. Une extension est obtenue en ajoutant à la formule un ensemble maximal consistant de littéraux. Ces extensions peuvent être calculés de manière itérative. Un programme logique est alors codé par un ensemble de clauses de logique propositionnelle. On prouve que chaque formule consistante admet au moins une extension et que, pour chaque modèle stable d'un programme logique, il existe une extension de son codage qui l'implique logiquement. En d'autres termes tout modèle stable est représenté par une extension. Certaines des extensions (les extra-extensions) ne correspondent pas à un modèle stable, mais sont intéressantes. Par exemple, en terme de représentation des connaissances, nous pouvons avoir un programme logique qui n'a pas de modèle stable à cause de la présence d'une seule règle. Mais l'ensemble des autres règles a des modèles stables pertinents. Dans ce cas les extra-extension permettent en fait d'isoler la règle gênante (on est proche de la para-consistance). D'autre part l'utilisation des extra-extensions va permettre d'avoir un algorithme simple de calcul d'extensions et de modèles stables.

Pour reconnaitre les extra-extensions, nous donnons une *condition discriminante*. L'ajout de cette condition permet, pour les programmes généraux d'avoir une bijection entre les extensions et les modèles stables. Enfin, nous décrivons un algorithme qui calcule les extensions de la formule CNF codant le programme logique. De cet ensemble d'extension, on peut extraire les modèles stables du programme logique initial. L'algorithme peut être mis en oeuvre avec un Solveur SAT légèrement modifié, qui effectue une énumération sur le codage CNF du programme logique et qui utilise un sous-ensemble de variables comme strong backdoor [16] (notée par STB). Ce strong backdoor donne la complexité de la procédure.

Le reste de l'article est structuré comme suit : dans la partie 2 nous donnons quelque préliminaires sur la programmation ASP. Nous introduisons la nouvelle sémantique des programmes logiques généraux et montrons sa relation avec la sémantique des modèles stables dans la partie 3. La partie 4 donne un algorithme basé sur une procédure de type DLL d'un solveurs SAT, qui permet de calculer les extensions à partir desquelles les modèles stables du programme logique peuvent être déduits. Nous donnons une conclusion et des perspectives dans la partie 5.

2 Notions de base

Cette partie rappelle quelques notions de base sur la programmation ASP. Il existe plusieurs classes de programmes logiques, caractérisées par la présence ou l'absence de la négation classique et de la négation par échec. Nous supposons dans la suite de ce travail que tous les programmes sont terminaux [1] (sans variables).

1. Chaque programme logique π est traduit en un programme logic terminal équivalent $ground(\pi)$ par la substitution de chaque règle contenant des variables par toutes ses instances

Les classes les plus connues sont les suivantes :

- *Les programmes logiques positifs :* un programme logique positif π est un ensemble de règles de la forme $r : L_0 \leftarrow L_1, \ldots, L_m$, $(m \geq 0)$ où L_i $(0 \leq i \leq m)$ est un atome. Un programme positif ne contient pas de négation classique ni de négation par échec. On peut le considérer comme un programme Prolog sans négation par échec ou aussi comme un ensemble de clauses de Horn du calcul propositionnel. Ici on a $head(r) = L_0$ et $body(r) = L_1, \ldots, L_m$. Le sens de la règle est "si on peut prouver le corps de la règle $L_1, L_2, \ldots, L_m$, alors on prouve la tête L_0". Etant donné un ensemble d'atomes A, on dit que la règle r est applicable (active) dans A si $body(r) \subseteq A$. Un ensemble d'atomes A est fermé par rapport à un programme logique π si et seulement si pour toute règle $r \in \pi$, si $body(r) \subseteq A$, alors $head(r) \in A$. Un programme logique positif π contient un seul modèle de Herbrand canonique équivalent à l'unique modèle minimal de Herbrand que nous dénotons par $CM(\pi)$. Le modèle minimal de Herbrand de π est le plus petit ensemble d'atomes fermé par rapport à π. Formellement, pour un programme logic π et un ensemble d'atomes A, l'opérateur $T_\pi(A) = \{head(r)/r \in \pi, body(r) \subseteq A\}$ calcule tous les atomes qui peuvent être déduits de A par utilisation des règles de π. Maintenant, nous définissons la suite : T_π^0=$T_\pi(\emptyset)$, $T_\pi^{k+1} = T_\pi(T_\pi^k), \forall k \geq 0$. Le modèle minimal de Herbrand est le plus petit point fixe de T_π, c'est-à-dire $CM(\pi)$=$\bigcup_{k\geq 0} T_\pi^k$. Le modèle minimal de Herbrand $CM(\pi)$ contient tous les atomes qui peuvent être déduits de π. Ce modèle est égal au modèle minimal exprimé par un programme Prolog formé par les règles de π ou à la partie positive du modèle minimal (au sens des modèles préférentiels) de l'ensemble de clauses de Horn correspondant.
- *Les programmes logiques généraux (normaux) :* un programme logique général π est un ensemble de règles de la forme $r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n$, $(0 \leq m < n)$ où L_i $(0 \leq i \leq n)$ sont des atomes, et *not* le symbole exprimant la négation par échec. Le corps positif de r est dénoté par $body^+(r) = \{L_1, L_2, \ldots, L_m\}$, et le corps négatif par $body^-(r) = \{L_{m+1}, \ldots, L_n\}$. Le mot *général* est du au fait que les règles sont plus générales que les clauses de Horn, car elles contiennent des négations par échec. La sous-règle $r^+ : L_0 \leftarrow L_1, L_2, \ldots, L_m$ exprime la projection positive de la règle r. Intuitivement, la règle r est interprétée par "*Si on infère tous les atomes* $\{L_1, L_2, \ldots, L_m\}$ *et qu'aucun des atomes* $\{L_{m+1}, \ldots, L_n\}$ *n'est inféré , alors on infère* L_0". Etant donné un ensemble d'atomes A, on dit que la règle r est applicable (active) dans A si $body^+(r) \subseteq A$ et $body^-(r) \cap A = \emptyset$. Le réduit du programme π par rapport à l'ensemble d'atomes A est le programme positif π^A où on supprime chaque règle contenant une expression $notL_i$ dans son corps négatif body telle que $L_i \in A$ et où on supprime les autres expressions $notL_i$ des corps des autres règles. Plus précisément, $\pi^A = \{r^+/r \in \pi, body^-(r) \cap A = \emptyset\}$. La sémantique la plus connue pour les programmes généraux est celle des modèles stables définie dans [7]. Cette dernière peut être vue comme une amélioration de la négation par échec de Prolog. Un ensemble d'atomes A est un modèle stable (un ensemble réponse) de π si et seulement si A est identique au modèle minimal de Herbrand de π^A, c'est à dire ssi $A = CM(\pi^A)$. La sémantique des modèles stables est basée sur l'hypothèse du monde clos, un atome qui n'est pas dans le modèle stable A est considéré comme faux.
- *Les programmes logiques étendus :* un programme logique étendu π est un ensemble de règles de la forme $r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n$ $(0 \leq m < n)$ où L_i, $(0 \leq i \leq n)$ sont des littéraux (des atomes L_i ou leurs négations $\neg L_i$). La sémantique des programmes logiques étendus [8] est une extension de la sémantique des modèles stables définie pour les programme logiques généraux [7]. Premièrement, considérons un programme logique étendu π qui ne contient pas de négations par échec (m=n, dans chaque règle de π). L'ensemble réponse de π est le plus petit ensemble de littéraux A tel que pour chaque règle $L_0 \leftarrow L_1, \ldots, L_m$ de π, si $L_1, \ldots, L_m$ sont dans A, alors L_0 est dans A, et si deux littéraux complémentaires figurent dans A, alors A devrait contenir tous les littéraux du langage de π. Nous dénotons cet ensemble réponse par $A = \alpha(\pi)$. Il est facile de voir qu'il existe un seul modèle $\alpha(\pi)$, et si π ne contient pas de négation, alors le modèle $\alpha(\pi)$ est identique au modèle minimal de Herbrand de π. Soient π un programme logique étendu et Lit l'ensemble de tous les littéraux de son langage. Pour tout sous-ensemble A de littéraux, le réduit du programme π par rapport à A est le programme π^A où on supprime chaque règle contenant une expression $notL_i$ dans son corps négatif telle que $L_i \in A$ et où on supprime les autres expressions $notL_i$ dans les corps des autres règles. Le programme résultant π^A ne contient pas de *not*, et le sous-ensemble A

terminales de telle sorte que chaque littéral de $ground(\pi)$ est terminal.

est un modèle stable pour π si et seulement si il est identique à l'ensemble réponse de π^A. C'est-à-dire, $A = \alpha(\pi^A)$. Ici, la sémantique est différente de celle des programmes généraux, puisque l'hypothèse du monde clos n'est pas appliquée dans le cadre des programmes étendus.

3 La négation par échec exprimée en logique propositionnelle et la notion d'extension

Notre sémantique, inspirée par celle introduite dans la logique des hypothèses [14, 13], part d'une approche logique classique propositionnelle non modale. Elle est basée sur un langage propositionnel L. Dans L on distingue deux types de variables propositionnelles : un sous ensemble de variables classiques $V = \{L_i : L_i \in L\}$ et un autre sous-ensemble $nV = \{notL_i : notL_i \in L\}$. Pour chaque variable dans $L_i \in V$ il existe une variable correspondante $notL_i \in nV$. L'ensemble total des variables de L est $V \cup nV$. Nous utilisons la variable propositionelle $notL_i$ pour exprimer la négation par échec utilisée dans les programmes logiques. A ce niveau, les variables L_i et $notL_i$ sont logiquement indépendants ; il n y a pas de relation sémantique entre ces variables.

Depuis l'introduction de Prolog en 1972 et sa négation par échec, le but de la programmation logique est de définir un lien entre les deux types de variables. Particulièrement, la logique des défauts [12], où la logique des hypothèses [14, 13] représentent deux façons de formaliser un tel lien. Généralement, toutes les logiques non-monotones oeuvrent dans ce sens.

Ici, dans le même esprit que la logique des hypothèses, nous introduisons une sémantique qui donne un lien entre les deux types de variables dans le cadre des ASP. Ce lien est exprimé par l'ajout au langage propositionnel L de l'ensembles de clauses négatives ME=$\{(\neg L_i \vee \neg notL_i) : L_i \in V\}$ équivalent à chacun des ensembles de formules $\{\neg(L_i \wedge notL_i) : L_i \in V\}$, $\{(L_i \rightarrow \neg notL_i) : L_i \in V\}$ et $\{(notL_i \rightarrow \neg L_i) : L_i \in V\}$. C'est en fait un pseudo axiome logique qui s'applique seulement aux couples de variables $\{L_i, notL_i\}$. Il est important de voir que l'ensemble de clauses ME ainsi généré exprime seulement les exclusions mutuelles entre chaque littéral $L_i \in V$ et son littéral négatif correspondent $notL_i \in nV$, mais n'établit pas l'équivalence entre $\neg L_i$ et $notL_i$. Ici, on peut avoir $\neg notL_i$ sans avoir L_i. Intuitivement, dans une première approche, on peut donc considérer qu'une variable $notL_i$ est une négation par échec classique affaiblie. Pour obtenir une vraie négation par échec, il suffit d'ajouter au système logique une propriété (en fait une règle d'inférence) qui oblige d'inférer L_i quand on infère $\neg notL_i$. Cette propriété sera appelée propriété discriminante dans la suite.

Etant donné un ensemble de formules F exprimées dans le langage propositionnel L et un sous-ensemble S de nV, on définit une extension de $(F \cup ME, S)$ comme la théorie obtenue de $F \cup ME$ en ajoutant un nombre maximal de littéraux $notL_i$ de S à $F \cup ME$ de telle sorte que la théorie résultante reste consistante. Une extension de $(F \cup ME, S)$ est donc une théorie maximalement consistante par rapport à l'inclusion de littéraux $notL_i$ de S. Formellement :

Définition 1 *Etant donné un ensemble de formules F du langage L, un sous-ensemble S de nV et un sous-ensemble S' de S, une extension de $(F \cup ME, S)$ est un ensemble $E = (F \cup ME) \cup S'$ tel que les conditions suivantes sont vérifiées :*

1. *E est consistant.*
2. *$\forall notL_i \in S - S', E \cup \{notL_i\}$ is inconsistant*

Exemple 1 *Soit $F = \{(notb \wedge c) \rightarrow a, a \rightarrow b, notd \rightarrow c, a\}$ un ensemble de formules du langage L, alors $ME = \{\neg a \vee \neg nota, \neg b \vee \neg notb, \neg c \vee \neg notc, \neg d \vee \neg notd\}$ et $S = \{notb, notd\}$ un sous-ensemble variables de nV. La paire $(F \cup ME, S)$ admet une seule extension $E = (F \cup ME) \cup \{notd\}$.*

Exemple 2 *Soit $F = \{(notb \wedge c) \rightarrow a, a \rightarrow b, notd \rightarrow c\}$ un ensemble de formules du langage L, on obtient $ME = \{\neg a \vee \neg nota, \neg b \vee \neg notb, \neg c \vee \neg notc, \neg d \vee \neg notd\}$, et $S = \{notb, notd\}$ un sous ensemble de variables de nV. La paire $(F \cup ME, S)$ admets deux extensions $E = (F \cup ME) \cup \{notd\}$ et $E = (F \cup ME) \cup \{notb\}$.*

On montre qu'un ensemble de formules de L contenant ME, qui est consistant a au moins une extension pour tout sous-ensemble $S \subset nV$.

Proposition 1 *Soient F un ensemble de formules du langage L et S un sous-ensemble de nV ($S \subset nV$). Si $(F \cup ME)$ est consistant alors il existe alors au moins une extension de $(F \cup ME, S)$.*

Preuve 1 *Soit S un sous-ensemble nV. Deux cas sont possibles : Si S est fini, alors la preuve est triviale. En effet, puisque $(E \cup ME)$ est consistant par hypothèse, alors en ajoutant successivement des littéraux $notL_i$ à $(E \cup ME)$, nous allons atteindre l'ensemble maximallement consistant $(E \cup ME) \cup S'$ où $S' \subset S$. Si S est infini, la preuve peut être faite par l'utilisation d'une part du théorème de compacité et d'autre part du lemme de Zorn (la preuve n'est pas donnée par manque de place).*

Proposition 2 *Si F est un ensemble de clauses dont chacune contient au moins un littéral positif de V et qui ne contiennent aucun littéral positif $notL_i$ de nV, alors l'ensemble de clauses $F \cup ME$ est consistant.*

Preuve 2 *L'interprétation composée de tous littéraux positifs $L_i \in V$ et de tous les littéraux $\neg notL_i \in nV$ est un modèle trivial.*

3.1 Une nouvelle sémantique pour la programmation logique

Dans la suite nous allons nous intéresser à la classe des programmes logiques généraux où un programme π est un ensemble de règles de la forme $r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n$, $(0 \leq m < n)$ et où tout L_i $(0 \leq i \leq n)$ est un atome (une variable propositionnelle). On considère que l'ensemble STB des littéraux positifs du type $notL_i$ qui apparaissent dans π, $STB = \{notL_i : notL_i \in \pi\} \subset nV$, est un strong backdoor [16] de π. Dans cette approche chaque règle r de π est traduite par une formule propositionnelle (une clause) $C = L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \ldots \neg notL_n$. On ajoute à cet ensemble de clauses traduites, l'ensemble des clauses d'exclusion mutuelle ME=$\{(\neg L_i \vee \neg notL_i) : L_i \in V\}$.

Un programme logique général π=$\{r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n\}$, $(0 \leq m < n)$, est donc traduit par un ensemble $L(\pi)$ de clauses de Horn propositionnelles qui représente son codage CNF dans un langage propositionnel L :

$$L(\pi) = \{\bigcup_{r \in \pi}(L_0 \vee \neg L_1 \vee, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \ldots \neg notL_n)$$
$$\bigcup_{L_i \in V}(\neg L_i \vee \neg notL_i)\}$$

La taille du codage $L(\pi)$ est donc égale à $taille(\pi) + 2n$, si n est le nombre de variables propositionnelles de π.

Soit donc $L(\pi)$ le codage en calcul propositionnel du programme logique π et $STB \subset nV$ son strong backdoor. Nous allons nous intéresser dans la suite aux extensions du couple $(L(\pi), STB)$. Une extension de $(L(\pi), STB)$ est la théorie obtenue à partir de $L(\pi)$ en ajoutant un ensemble maximal consistant de littéraux $notL_i \in STB$ to $L(\pi)$ tel que la théorie obtenue soit consistante. A partir de la définition **??** on définit donc un cas particulier d'extensions pour les programmes logiques :

Définition 2 *Soit $L(\pi)$ le codage en logique d'un programme logique π, STB son strong backdoor, et un sous ensemble $S' \subset STB$. Alors $E = L(\pi) \cup S'$ est une extension de $(L(\pi), STB)$ si les conditions suivantes sont vérifiées :*

1. *E est consistant*
2. *$\forall notL_i \in STB - S', E \cup \{notL_i\}$ est inconsistant.*

Exemple 3 *On considère l'ensemble F de formules propositionnelles données dans l'exemple 1 comme la traduction logique d'un ensemble de règles π. On donne ci dessous π et son codage logique $L(\pi) = L(\pi) - Rules \cup L(\pi) - ME$:*

π :	$L(\pi) - Rules$:	$L(\pi) - ME$:
$a \leftarrow c, notb$	$a \vee \neg c \vee \neg notb$	$\neg a \vee \neg nota$
$b \leftarrow a$	$b \vee \neg a$	$\neg b \vee \neg notb$
$c \leftarrow notd$	$c \vee \neg notd$	$\neg c \vee \neg notc$
$a \leftarrow$	a	$\neg d \vee \neg notd$

Le strong backdoor est alors l'ensemble $STB = \{notb, notd\}$ et la paire $(L(\pi), STB)$ a une seule extension $E = L(\pi) \cup \{notd\}$. En effet E est consistant (il existe un modèle) et est maximal consistant sur le STB (si on ajoute $notb$ à E, l'ensemble obtenu est inconsistant).

Exemple 4 *On prend π le programme logique de l'exemple 3 vu ci dessus et on supprime la règle $a \leftarrow$. On obtient le programme logique $\pi' = \pi - \{a \leftarrow\}$ dont le codage logique CNF est $L(\pi') = L(\pi) - \{a\}$ et dont le STB est le même que celui de π. Dans ce cas on obtient deux extensions de $(L(\pi'), STB)$: $E_1 = L(\pi') \cup \{notd\}$ et $E_2 = L(\pi) \cup \{notb\}$. Mais $(L(\pi')$ n'a pas de modèle stable. Les deux extensions sont des extra-extensions dont on rediscutera dans la suite.*

Nous allons maintenant montrer que tout modèle stable correspond à une extension.

Théoreme 1 *Si X est un modèle stable d'un programme logique π, alors il existe une extension E de $(L(\pi), STB)$ telle que $X = \{L_i \in V : E \models L_i\}$. D'autre part, E vérifie la "condition discriminante" : $(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$.*

Preuve 3 *Soit l'ensemble $E = L(\pi) \cup \{notL_i / L_i \notin X\}$. Pour démontrer le théorème on montre que E est une extension (1 and 4 ci dessous), que $X = \{L_i : E \models L_i\}$ (2 and 3) et que E vérifie la condition discriminante $(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$ (5).*

1. *E est consistant : on montre que l'interprétation $I = \{\neg L_i, notL_i / L_i \notin X\} \cup \{L_i, \neg notL_i / L_i \in X\}$ est un modèle de E. Comme $\{notL_i / L_i \notin X\} \subset I$, il suffit de montrer que I est un modèle de $L(\pi)$. On doit montrer que chaque clause $(\neg L_i \vee \neg notL_i)$ de ME et chaque clause de $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \cdots \vee \neg notL_i \cdots \vee \neg notL_n)$ de $L(\pi)$ qui correspond à une règle de π est satisfaite par I. Il est évident que chaque clause $(\neg L_i \vee \neg notL_i)$ de ME est satisfaite par I. En effet, si $L_i \in X$, on a $\neg notL_i \in I$ par définition de I et la clause est satisfaite par I. Maintenant, si $L_i \notin X$, alors par définition de I on a $\neg L_i \in I$, donc I satisfait la clause $(\neg L_i \vee \neg notL_i)$. Donc $(\neg L_i \vee \neg notL_i)$ est satisfaite par I dans les deux cas. Maintenant pour prouver que chaque clause $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \cdots \vee \neg notL_i \cdots \vee \neg notL_n)$ est satisfaite par I, on étudie aussi deux cas. Si la sous clause $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m)$ correspond à une regle du réduit π^X, alors X satisfait $L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m$ puisque X est un modèle stable π,*

alors I satisfait la clause $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m)$ *car* $X \subset I$. *Donc la clause* $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \cdots \vee \neg notL_i \cdots \vee \neg notL_n)$ *est également satisfaite par I. Dans l'autre cas, la sous clause* $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m)$ *ne correspond pas à une regle de* π^X. *Il y a alors un* $notL_i$ *dans la partie négative de la regle de* π *qui correspond à la clause* $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \cdots \vee \neg notL_i \cdots \vee \neg notL_n)$ *of* $L(\pi)$ *tel que* $L_i \in X$. *Alors,* $\neg notL_i \in I$ *par définition de I et notre clause est satisfaite par I. I est donc un modele de* $L(\pi)$ *et de E, donc E est consistant.*

2. $L_i \in X \Rightarrow E \models L_i$ **:** *à partir de l'ensemble X on obtient le réduit* π^X *en enlevant en premier lieu dans* π *toute règle* $r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n$ $(0 \leq m < n)$ *qui contient un littéral* $notL_i$ *dans sa partie négative, tel que* $L_i \in X$. *Dans ce cas la clause correspondante* $c : L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \ldots \neg notL_n$ *de l'ensemble E n'est pas supprimée. Ensuite on supprime dans les autres règles de* π *toutes les occurrences de* $notL_i$ *tel que* $L_i \notin X$, *et dans ce cas, on en déduit à partir de la définition de E que* $notL_i \in E$, *puis on supprime les littéraux* $\neg notL_i$ *dans les clauses correspondantes de E. On en déduit que le réduit* π^X *est inclus dans l'ensemble* E^X *obtenu à partir de E en appliquant la résolution unitaire sur les littéraux de la forme* $notL_i$ $(E \models E^X)$. *Comme X est un modèle stable de* π, *par définition,* L_i *appartient au modèle de Herbrand minimal de* π^X. *On en déduit que* $\forall L_i \in X, \pi^X \models L_i$. *Comme* $\pi^X \subset E^X$, *alors* $E^X \models \pi^X$ *et donc* $E \models \pi^X$. *Comme l'inférence est monotone on déduit que* $\forall L_i \in X, E \models L_i$.

3. $E \models L_i \Rightarrow L_i \in X$ **:** *c'est équivalent à montrer que* $L_i \notin X \Rightarrow E \not\models L_i$. *De* $L_i \notin X$ *on a par définition de E que* $notL_i \in E$, *donc par application de la unit résolution sur la clause* $(\neg L_i \vee \neg notL_i)$ *générée par le nouveau pseudo axiome, on déduit que* $E \models \neg L_i$. *On en conclu que* $E \not\models L_i$ *car E est consistant (montré en 1).*

4. *E est maximal consistant sur STB : Soit l'ensemble* $E' = E \cup \{notL_i\}$, *tel que* $notL_i \in STB$, *but* $notL_i \notin E$. *Comme* $notL_i \notin E$, *par définition de E, on déduit que* $L_i \in X$. *En appliquant la propriété démontrée en 2) on obtient* $E \models L_i$. *Donc,* $E' \models L_i$. *En appliquant la résolution unitaire sur la clause* $(\neg L_i \vee \neg notL_i)$ *générée par le pseudo axiome, on déduit que* $E' \models \neg notL_i$ *et que* E' *n'est pas consistant car il infère* $notL_i$ *et son complément* $\neg notL_i$. *Donc E est maximal consistant sur STB.*

5. $(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$ *: il est équivalent de montrer que* $(\forall L_i \in V, E \not\models L_i \Rightarrow E \not\models \neg notL_i)$. *Si* $E \not\models L_i$, *comme* $X = \{L_i : E \models L_i\}$, *on déduit que* $L_i \notin X$, *donc* $notL_i \in E$ *par définition de E. Comme E est consistant on conclut que* $E \not\models notL_i$.

Exemple 5 *Soit le programme logique* π *de l'exemple 3 et son codage CFN* $L(\pi)$. π *a un seul modèle stable* $X = \{a, b, c\}$ *qui correspond à l'unique extension* $E = L(\pi) \cup \{notd\}$ *de* $(L(\pi), STB)$. *En effet, après l'affectation des variables du STB, on utilise la résolution unitaire pour montrer que* $E \models \{a, b, c, \neg d, \neg nota, \neg notb, \neg notc\}$. *On voit ici que E vérifie bien la condition discriminante* $(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$ *et que* $X = \{L_i \in V : E \models L_i\} = \{a, b, c\}$.

Remarque 1 *Comme* (π) *est un programme général, une extension E de* $(L(\pi), STB)$ *est un ensemble consistant de clauses de Horn . Donc tous les littéraux positifs (les clauses unitaires positives) qui peuvent être inférées par E sont produits par résolution unitaire. Par exemple les littéraux du modèle stable* $X = \{a, b, c\}$ *de l'exemple précédent sont inférés par résolution unitaire. Ceci est important pour la complexité de l'algorithme de calcul des extensions et des modèles stable qui va être donné dans la partie 4.*

Remarque 2 *La proposition 1, dit que* $(L(\pi), STB)$ *a toujours une extension si* $L(\pi)$ *est consistant. Mais un programme logique* π *peut ne pas avoir de modèle stable. Il peut donc exister des extensions (extra-extensions) de* $(L(\pi), STB)$ *qui ne correspondent à aucun modèle stable de* π. *On va montrer que ces extra-extensions sont exactement celles qui ne vérifient pas la condition discriminante. Ces extra-extensions, non prises en compte par la sémantique des modèle stables, sont très intéressantes pour simplifier le calcul d'extension (donc de modèles stables). Elles sont également importantes pour la représentation des connaissances, mais on ne discutera pas de cet aspect dans cet article.*

Par exemple, dans l'exemple 4, on a deux extensions de $(L(\pi'), STB) : E_1 = L(\pi') \cup \{notd\}$ et $E_2 = L(\pi) \cup \{notb\}$ qui ne correspondent pas à un modèle stable. D'un autre coté, on peut montrer que $E_1 \models \{c, \neg d, \neg notb, \neg notc\}$ et que $E_2 \models \{\neg b, \neg a, \neg c, \neg notd\}$. On remarque donc que ces deux extensions ne vérifient pas la condition discriminante du Théorème 1, car E_1 implique $\neg notb$ mais n'implique pas b et également E_2 implique $\neg notd$ mais n'implique pas d. On parlera dans la suite d'extra-extensions.

Ci dessous on donne deux exemples d'école très utilisés par la communauté qui étudiait la logique des défauts à une époque. On montre que notre sémantique est valide pour les deux exemples .

Exemple 6 *Soit le programme* $\pi = \{a \leftarrow nota\}$ *composé dune seule règle. Ce programme n'a pas de modèle stable. Son codage CNF est* $L(\pi) = \{a \vee \neg nota, \neg a \vee \neg nota\}$ *et son strong backdoor est* $STB = \{nota\}$. *La paire* $(L(\pi), STB)$ *a une extra-extension* $E = \{\neg nota\}$. *En effet par résolution sur les deux clauses de* $L(\pi)$ *on infère la clause unitaire* $\neg nota$ *qui subsume les clauses de* $L(\pi)$. *Donc* $E \models \neg nota$ *et* $E \not\models a$. *L'extension E ne vérifie donc pas la condition discriminante. On prouve donc pour cet exemple que* $(L(\pi), STB)$ *a une seule extension qui est une extra-extension. Notre sémantique capture bien la sémantique classique des modèles stables pour cet exemple.*

Exemple 7 *On prend le programme logique donné dans l'exemple 6 auquel on ajoute une règle* $a \leftarrow$. *On obtient*

le programme $\pi = \{a \leftarrow nota, a \leftarrow\}$ *qui a un unique modèle stable* $X = \{a\}$. *Son codage CNF est* $L(\pi) = \{a \vee \neg nota, \neg a \vee \neg nota, a\}$ *et son strong backdoor est* $STB = \{nota\}$. *La paire* $(L(\pi), STB)$ *a une extension* $E = L(\pi)$, *telle que* $E \models \neg nota$ *et* $E \models a$. *Cette extension vérifie bien la condition discriminante et implique le modèle stable* $X = \{a\}$ *de* π. *La sémantique capture bien ici aussi les modèles stables.*

Nous allons maintenant généraliser les exemples ci dessus en prouvant que toute extension de $(L(\pi), STB)$ qui vérifie la condition discriminante implique un modèle stable de π. On aura donc une bijection entre les modèles stables et les extensions qui ne sont pas des extra-extensions

Théoreme 2 *Si* E *est une extension de* $(L(\pi), STB)$, *qui vérifie la condition* $(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$ *alors then* $X = \{L_i : E \models L_i\}$ *est un modèle stable de* π.

Preuve 4 *Soit* E *une extension de* $(L(\pi), STB)$. *Il existe donc un ensemble* $S' \subset STB$ *tel que* $E = L(\pi) \cup S'$. *Comme* E *est maximal consistant sur* STB, *on a* $E \models \neg notL_i$ *pour tout* $notL_i \in STB - S'$. *Donc,* $E \models L(\pi) \cup S' \cup \{\neg notL_i : notL_i \in STB - S\}$. *D'un autre coté l'ensemble* $X = \{L_i : E \models L_i\}$ *est un modèle minimal de* E. *En utilisant le pseudo axiome et la condition discriminante on montre que* $X = \{L_i : E \models L_i\} = \{L_i : E \models \neg notL_i\}$. *Pour prouver que* X *est un modèle stable* π, *on doit montrer que* X *est un modèle minimal de* π^X. *Pour ce faire on étudie deux cas. Pour le premier cas, si* $notL_i \in E$, *on supprime toute les occurrences du littéral* $\neg notL_i$ *dans toutes les clauses* c : $(L_0 \vee \neg L_1 \vee \neg L_2, \ldots, \neg \vee L_m \vee \neg notL_{m+1} \ldots, \neg notL_i, \ldots, \neg notL_n)$ *de* $L(\pi)$. *En utilisant la clause* $(\neg L_i \vee \neg notL_i)$ *de* ME *on déduit que* $E \models \neg L_i$, *et donc que* $E \not\models L_i$ *car* E *est consistant. On en déduit que* $L_i \notin X$, *et dans ce cas, pour obtenir le réduit* π^X, *on supprime toutes les occurrences de* $notL_i$ *dans le corps de la règle* r : $L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_i, \ldots, notL_n$ *de* π *qui correspond à la clause c. Pour le second cas* $notL_i \notin E$, *et donc* $E \models \neg notL_i$ *et toute clause c de* $L(\pi)$ *qui contient un littéral* $\neg notL_i$ *est subsumée dans E. En utilisant la condition discriminante E, on déduit que* $E \models L_i$. *Donc* $L_i \in X$, *et dans ce cas, pour obtenir la réduction* π^X, *la règle correspondante r dans* π *de la clause c est supprimée. On en conclut que la simplification de E par les littéraux du STB donne une réduction* π^X *qui est une conséquence logique de E. Donc* $E \models \pi^X$. *Comme* X *est un modèle minimal de E, on a que* X *est un modèle minimal de* π^X. *Par définition des modèles stables, on conclut que* X *est un modèle stable de* π.

On donne ci dessous deux autres exemples classiques qui viennent de la logique des défauts.

Exemple 8 *Soit le programme logique* π *composé de deux règles :*

$$\pi = \{a \leftarrow notb, b \leftarrow nota\}$$

Son codage CNF est l'ensemble de clauses $L(\pi) = Rules \cup ME$ *:*

$$Rules = \{a \vee \neg notb, b \vee \neg nota\}$$

$$ME = \{\neg a \vee \neg nota, \neg b \vee \neg notb\}$$

Son backdoor est $STB = \{nota, notb\}$. *La paire* $(L(\pi), STB)$ *a deux extensions* $E_1 = L(\pi) \cup \{nota\}$ *and* $E_2 = L(\pi) \cup \{notb\}$. *Par résolution unitaire on déduit que* $E_1 \models \{b, \neg a, \neg notb\}$ *and* $E_2 \models \{a, \neg b, \neg nota\}$. *Donc ces extensions vérifient la condition discriminante. A partir de* E_1 *resp.* E_2 *on obtient les ensembles de littéraux positifs impliqués* $X_1 = \{b\}$ *resp.* $X_1 = \{a\}$ *qui sont bien les deux modèles stables du programme* π.

Exemple 9 *Soit le programme logique* π *:*

$$\pi = \{a \leftarrow notb, b \leftarrow notc, c \leftarrow nota\}$$

Son codage CNF est l'ensemble de clauses $L(\pi) = Rules \cup ME$ *:*

$$Rules = \{a \vee \neg notb, b \vee \neg notc, c \vee \neg nota\}$$

$$ME = \{\neg a \vee \neg nota, \neg b \vee \neg notb, \neg c \vee \neg notc\}$$

,

Le backdoor est $STB = \{nota, notb, notc\}$. *La paire* $(L(\pi), STB)$ *a trois extensions* $E_1 = L(\pi) \cup \{nota\}$, $E_2 = L(\pi) \cup \{notb\}$ *et* $E_3 = L(\pi) \cup \{notc\}$. *Par résolution unitaire, on déduit que* $E_1 \models \{\neg a, c, \neg notc, \neg notb\}$, $E_2 \models \{\neg b, a, \neg nota, \neg notc\}$ *et* $E_3 \models \{\neg c, b, \neg notb, \neg nota\}$. *On voit que les trois extensions sont des extra-extensions qui ne vérifient pas la condition* $(\forall L_i \in V, E \models \neg notL_i \Rightarrow E \models L_i)$. *Donc le théorème 2 dit que* π *n'a pas de modèle stable, ce qui est bien le cas.*

On a donc montré avec les théorème 1 et théorème 2 que les modèles stables d'un programme logique π sont en bijection avec un sous ensemble des extensions de $(L(\pi), STB)$. Nous avons aussi caractérisé les extra-extensions par une condition simple à vérifier. D'autre part on a vu qu'un ensemble de formules consistant a toujours une extension. Cette dernière propriété entraine que la définition d'extension n'est pas une définition par point fixe et qu'il est possible de calculer les extensions de manière incrémentale : on ajoute des $notL_i$ du STB en vérifiant à chaque ajout la consistance de l'ensemble obtenu. On peut donc obtenir des algorithmes très simples de calcul d'extension. Si on ne s'intéresse pas aux extra-extension, il suffit de les filtrer à la fin du calcul : dès qu'une extension a été trouvée on teste si elle correspond un modèle stable en vérifiant la propriété discriminante. On va donner dans la partie suivante une méthode de calcul d'extensions et de modèles stables basée sur ces considérations.

4 Un algorithme de calcul d'extensions ou de modèles stables basé sur la nouvelle sémantique

On sait que tout modèle stable d'un programme π est un modèle de sa complétion $comp(\pi)$, mais que la réciproque est fausse dans le cas général. Fages [6] montre que si un programme π est " tight" (sans boucle) alors l'ensemble des modèles stables est égal à l'ensemble des modèles de sa

complétion $comp(\pi)$ [1]. Si la complétion d'un programme sans boucle est traduite par un ensemble de clauses Γ, alors on peut utiliser un solveur SAT Γ comme une boite noire pour générer les modèles stables de π. Lin et Zhao [3] ont montré que pour les programmes avec boucle, les modèles de la complétion qui ne sont pas des modèles stables peuvent être éliminés en ajoutant à la complétion des *formules boucles*. Leur solveur ASSAT est basé sur cette technique et est plus performant que les solveurs ASP classiques tel que Smodels [11, 15] et DLV [4] pour plusieurs instances. Néanmoins le solveur ASSAT a quelques défauts : il ne peut calculer qu'un seul modèle stable et la formule calculée peut exploser de manière combinatoore en espace. En prenant en compte le fait que chaque modèle stable d'un programme π est un modèle de sa complétion $comp(\pi)$, Guinchiglia et al. in [9] n'utilisent pas un solveur SAT comme boite noire, mais implêmentent une méthode basée sur la procédure DLL [2] dans laquelle ils introduisent une fonction qui vérifie si un modèle généré est un modèle stable. Cette méthode a été implémentée dans le système Cmodels-2 [10] et a l'avantage de calculer $comp(\pi)$ sans introduire de variable supplémentaire, autre que celles utilisées par la transformation en clauses de $comp(\pi)$. Elle fonctionne également pour les programmes avec boucles et sans boucles.

4.1 La méthode

La méthode décrite ici utilise également un soldeur SAT mais elle est différente car elle basée sur la nouvelle sémantique décrite plus haut. Pour un programme π la méthode travaille sur $L(\pi)$ et non pas sur la complétion $comp(\pi)$. Si π est un programme général, $L(\pi)$ est un ensemble de clauses de Horn construit sur deux ensembles de variables propositionnelles V and nV. L'ensemble $STB \subset nV$ des variables qui apparaissent dans le corps des règles de π est un strong backdoor [16] que l'on énumère pour obtenir les extensions de $(L(\pi), STB)$ à partir desquelles on peut trouver les modèles stables de π.

Proposition 3 *Soit π un programme général et $L(\pi)$ son codage CNF. Soit E une extension de $L(\pi)$sur laquelle on a appliqué la simplification par résolution unitaire et subsomption (suppression de clauses par subsomption et suppression de variables propositionnelles par résolution unitaire). On a alors les propriétés suivantes :*

1. *L'ensemble de clauses E est l'union d'un ensemble C_1 de clauses unitaires C_1 et d'un ensemble C_2 de clauses de Horn non unitaires qui ne contient pas de variables propositionnelles du STB. De plus les ensembles C_1 and C_2 n'ont pas de variable en commun (variables indépendants).*
2. *Si on affecte à faux tous les $\{notL_i\}$ et tous les $\{L_i\}$ qui ne sont pas déjà affectés dans C_1 on obtient le modèle minimal M de E (il est unique car E est un ensemble de clauses de Horn). Ce modèle minimal sera d'une part la partie positive de l'extension E (l'ensemble des littéraux positifs impliqué par E) et d'autre part un candidat comme modèle stable.*
3. *Pour savoir si M est un modèle stable, il suffit de vérifier si la condition caractéristique est vérifiée. Comme toutes les variables ont été affectées, cette vérification est immédiate.*

Preuve 5
- *Assertion 1 : $L(\pi)$ est un ensemble de clauses de Horn. Donc après l'affectation de toutes les variables du backdoor STB et la simplification par résolution unitaire et subsomption, le codage $L(\pi)$ est l'union de l'ensemble des clauses unitaires obtenues par la résolution unitaire et d'un ensemble de clauses de Horn non unitaires. Le fait que ces ensembles n'aient pas de variable propositionnelle en commun vient du fait que $L(\pi)$ a été simplifié par résolution unitaire et subsomption.*
- *Assertion 2 : Après affectation des $\{notL_i\}$ du STB, simplification par résolution unitaire et subsompsion il ne reste plus de $\{notL_i\}$ du STB dans les clauses non unitaires de E. On peut donc compléter l'interprétation en affectant à faux les variables propositionnelles non affectées. On obtient alors un modèle minimal (au sens des modèles préférentiels) qui est un candidat à la vérification pour être modèle stable.*
- *Assertion 3 : Ceci a été montré dans la démonstration du théoreme 2.*

A partir de ce qui précède on va donner ci dessous les étapes principales d'une méthode de calcul de toutes les d'extension et de tous les modèles stables.

1. Soit $E = L(\pi)$.
2. On affecte les variables du STB en utilisant une méthode d'énumération DLL. De plus on vérifie à chaque affectation la consistance. Comme on a un ensemble de clauses de Horn, une propagation par résolution unitaire assure la consistance. On maintient également à chaque point de choix l'ensemble des littéraux $\{\neg notL_i\}$ impliquées par l'état courant. Pour les maintenir les $\{\neg notL_i\}$ impliqués, une méthode très brutale est de faire k tests de consistance (k propagations unitaire) si k est le cardinal du STB (on peut être bien plus efficaces). Pour la stratégie d'affectation des variables du STB, une bonne approche est d'affecter en premier à Vrai les variables libres du STB. Puis on ne reviendra en arrière que sur les les noeuds qui ont inféré au moins un nouveau littéral négatif du STB. Cette stratégie assure qu'à chaque point de choix consistant, en poursuivant l'énumération du STB, on obtiendra au moins une extension (la maximalité de l'extension est donc obtenue automatiquement).
   Une fois l'affectation du STB effectuée (on est sur une feuille de l'arbre de recherche restreint au STB), on obtient donc une interprétation partielle I_{STB} du STB qui étend E ($E = E \cup I_{STB}$)).
3. On affecte à Faux toutes les variables de E non affectées pour obtenir le modèle minimal M de E.
4. Si la condition ($\forall L_i \in V : E \models \neg notL_i \Rightarrow E \models L_i$) est vérifié, la restriction aux littéraux positifs de E est un modèle stable de π. Cette vérification est

immédiate à partir du modèle minimal M obtenu à l'étape 3.

L'algorithme est décrit ici de manière très générale. Dans la pratique on peut le rendre bien plus efficace, mais ce n'était pas le premier but de cet article.

4.2 Complexité

Si n est le nombre de variables du codage $L(\pi)$ d'un programme π, k est le cardinal du STB , m est le nombre de clauses de $L(\pi)$, l'étape 2 de l'algorithme peut être effectuée en $O(knm2^k)$. Donc la complexité asymptotique dans le pire des cas est $O(knm2^k)$ avec $k \leq n$. Cette estimation est très grossière et peut être très améliorée. En particulier avec des considérations simples on peut diviser k par 2. Dans le cas moyen la complexité est bien moindre. De plus, il est intéressant de remarquer que la première extension est trouvée à la première descente dans l'arbre sans retour en arrière, donc en $O(knm)$. Enfin la méthode a l'avantage de travailler sur un codage CNF en espace constant $L(\pi)$ dont la taille est $taille(\pi) + 2n$. Cette méthode peut être utilisée pour des programmes avec boucles et sans boucles. Elle est capable de calculer tous les modèles stables d'un programme logique général π. La méthode peut être implémentée, avec une modification mineure de la procédure DDL. Ce travail est théorique mais tous les outils pour l'implémenter existent. On pourra alors faire les comparaisons de performance avec les méthodes classiques

5 Conclusion

Nous avons décrit, de manière théorique, une nouvelle sémantique pour la programmation logique. Cette sémantique n'est pas une sémantique de point fixe classique. Les règles d'un programme logique sont codées par un ensemble de clauses propositionnelles dans un cadre de logique classique. On définit, de manière générale, une notion d'extension qui appliquée aux programmes logiques permet de capturer les modèles stables. On montre que chaque programme logique consistant a au moins une extension et que tout modèle stable d'un programme logique est impliqué par une extension. La notion d'extension est plus générale que les modèles stables et on a des extra-extensions qui ne représentent pas un modèle stable. On donne un critère simple pour reconnaitre ces extra-extensions. Enfin on définit un algorithme qui permet de calculer les extensions et les modèles stables. Cet algorithme est basé sur une modification mineure de la procédure DDL de SAT et énumère un ensemble restreint STB de variables propositionnelles, dont dépend la complexité.

Comme travail futur nous nous intéresserons à l'implémentation de l'algorithme et à sa comparaison avec les méthodes existantes. D'autre part le formalisme des extensions peut facilement prendre en compte les programmes étendus avec disjonctions en partie gauche des règles : au lieu de fournir un unique modèle minimal, une extension en fournira plusieurs. Ceci coûte bien entendu une augmentation de la la complexité algorithmique (on quitte les clauses de Horn). Pour étudier les programmes étendus la notion d'extension et les techniques décrites ici fournissent une bonne base. On peut aussi se rapprocher de la logique des défauts et des logiques non monotones plus subtiles, mais pour ceci il faut revenir un peu vers la logique des hypothèses.

Références

[1] K. Clark. Negation as failure. In *Logic and data bases*, pages 293–322. In Herve Gallaire and Jack Minker, editors, 1978.

[2] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *JACM, 5(7).*

[3] E.Erdem and Y.Zhao. Assat : Computing answer sets of a logic program by sat solver. In *AAAI-02*, 2002.

[4] T. Eiter. Th kr sysem dlv :progress report, comparisons and benchmarks. In *KR*, 1978.

[5] E. Erdem and V. Lifschitz. Tight logicprograms. *Thoery and Practice of Logic Programming*, 3 :499–518, 2003.

[6] F. Fages. Consistency ofclark's completion and existence of stable models. *Thoery and Practice of Logic Programming*, 1 :51–60, 1994.

[7] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic programming : Fifth Int'l Conf. and Symp.*, pages 1070–1080. In Robert Kawalski and Kenneth Bowen editors, 1988.

[8] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9 :365–385, 1991.

[9] E. Giunchiglia, Y. Lierler, and M. Mratea. Sat-based answer set programming. In *19th National Conference on Artificial Intelligence,July 25-29, San Jose, California. AAAI*, 2004.

[10] Yuliya Lierler and Marco Maratea. Cmodels-2 : Sat-based answer set solver enhanced to non-tight programs. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 346–350, 2004.

[11] I. Niemela. Logic programs with stable models semantics as a constraint programming paradigm. *Anals of mathematics and Artificial Intelligence*, 25 :241–273, 1999.

[12] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13 :81–132, 1980.

[13] C. Schwind and P. Siegl. A modal logic for hypothesis theory. *Fondamunta Informatika*, 21 (1/2) :89–101, 1994.

[14] P. Siegl and C. Schwind. Hypothesis theory for nonmonotonic reasoning. In *Workshop on Nonstandard Queries and Answers*, Toulouse, July 1991.

[15] P. Simons. Extending and implementing the stable model semantics. *In doctoral dessertation*, pages 305–316, 2000.

[16] Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *IJCAI*, pages 1173–1178, 2003.

Branch and Learn pour l'acquisition de CSP

Christian Bessiere[1] **Rémi Coletta**[1] **Frédéric Koriche**[1]
Arnaud Lallouet[2] **Matthieu Lopez**[3]

[1] LIRMM, Université de Montpellier, 34095 Montpellier Cedex 5
[2] GREYC, Université de Caen, 14032 Caen CEDEX
[3] LIFO, Université d'Orléans, F-45067 ORLEANS Cedex 2
{bessiere, coletta, koriche}@lirmm.fr arnaud.lallouet@unicaen.fr
matthieu.lopez@univ-orleans.fr

Résumé

L'utilisation de la programmation par contraintes, notamment la modélisation des problèmes, est devenue limitée à des utilisateurs possédant une bonne expérience dans le domaine. Ce papier s'inscrit dans un cadre visant à automatiser la modélisation. Les techniques existantes ont montré des résultats encourageants mais certaines exigences rendent leur utilisation encore problématique. Nous nous intéressons à la possibilité pour l'utilisateur de ne pas donner de solutions et de non-solution de son problème. En partant d'un CSP sans aucune contrainte, notre méthode consiste à résoudre le problème de l'utilisateur de manière classique en développant un arbre de recherche. Quand notre outil ne peut décider si l'affectation partielle courante est correcte ou non, nous demandons à l'utilisateur de guider la recherche sous forme de requêtes. Ces requêtes permettent de guider la recherche en répondant à des requêtes et de trouver des contraintes à ajouter aux modèles du CSP et ainsi améliorer la recherche.

Abstract

Constraint programming (CP) allows to model problem with efficient solving methods. However, its use is limited to CP expert users notably for the modeling step. This paper aims to partially automate this modeling part. Existing methods have shown encouraging results, but their uses are still complicated. We are interesting in simplifying this allowing the user to not give solutions and non-solutions of her problem. Beginning with a CSP without constraint, our method consists in solving the problem in a classical way. Developing the search tree, we progressively assign values to variables. When our tool cannot decide for itself if a partial assignment is correct or not, we ask, with a query, to the user to guide the search. These queries allow to learn constraint and then improve the solving.

1 Introduction

Il est classique de dire que la Programmation par Contraintes permet de résoudre efficacement un grand nombre de problèmes combinatoires avec une relative facilité de modélisation. Toutefois, cette facilité n'est qu'apparente car elle cache en fait un certain nombre de difficultés, la première desquelles [13] étant la modélisation. La modélisation, première étape de la résolution d'un problème, possède trois niveaux de difficulté. Le premier est de déterminer quelles sont les variables et leur domaine, aussi appelé "point de vue" [13]. Le second consiste à déterminer les contraintes du problème. Le troisième enfin consiste à reformuler l'ensemble pour adapter le modèle aux particularités du solveur qui servira à le résoudre. Il peut s'agir d'introduire de nouvelles variables, des contraintes redondantes [9], voire des modèles redondants [7] qui ne serviront qu'à accélérer le processus de résolution.

Traditionnellement, cette phase de modélisation est faite par l'utilisateur. Si obtenir un modèle performant d'un problème difficile nécessite une expertise considérable [12], on peut noter que de nombreux utilisateurs sont tout autant déroutés par l'écriture d'un modèle simple. Nous proposons d'assister l'utilisateur en lui proposant de l'aider à construire son modèle en répondant à une série de questions permettant la découverte des contraintes. On pourra noter qu'il existe des approches visant à assister l'utilisateur dans la construction du point de vue [6] ou la reformulation de son modèle [5].

La construction d'un CSP par apprentissage a été introduite avec le système CONACQ [8] qui construit un espace de version contenant tous les CSP possibles et le fait converger avec le traitement de solutions et de

non-solutions du problème. Toutefois, cette approche est limitée à la découverte du modèle car l'utilisateur se doit de posséder des solutions de son problème. Dans un cadre général, il est probable que la découverte d'une solution soit suffisante à satisfaire l'utilisateur. Afin de remédier à ce problème, une autre approche utilise des solutions et non-solutions de problèmes proches et réalise l'apprentissage grâce à des techniques de Programmation Logique Inductive [10].

Nous présentons ici une nouvelle technique basée sur Conacq mais ne nécessitant plus de connaitre à l'avance des solutions du problème à résoudre. Pour cela, en partant d'un CSP initial vide, nous parcourons un arbre de recherche destiné à résoudre le problème (encore inconnu) de l'utilisateur. A chaque fois que la validité d'une branche est inconnue, nous posons la question de sa validité, sous forme de requêtes d'appartenance avec instances partielles des variables, à l'utilisateur et ajoutons au CSP en cours de construction les contraintes que l'on peut déduire de sa réponse, tout comme le branch-and-bound ajoute des contraintes pour changer la validité de solutions sous-optimales. Nous appelons cette approche *branch-and-learn*. Bien entendu, le système utilise la connaissance acquise jusqu'ici afin de ne pas poser de question qui soit déductible de la propagation des contraintes déjà connues.

Nous présentons en détail le système Conacq dans la section 2, puis en section 3 l'extension de Conacq aux requêtes partielles nécessaires pour pouvoir poser des questions au cours de la création de l'arbre de recherche. Dans la section 4, l'algorithme de génération de requêtes parcourant l'arbre de recherche est présenté comme un générateur d'exemples pour l'apprentissage, et nous terminons par une évaluation des résultats sur notre implantation.

2 CONACQ et la génération de requêtes

Dans un premier temps, nous commençons par décrire CONACQ, un système visant à acquérir grâce à des exemples étiquetés, les contraintes d'un problème. Nous débuterons en donnant une intuition du fonctionnement du système qui est basé sur l'espace des versions. Nous décrirons comment l'approche est encodée dans un problème SAT et la manière dont les exemples sont traités. Enfin, nous verrons comment étendre CONACQ à l'utilisation de requêtes partielles.

2.1 Acquisition de contraintes, espace des versions et formulation par un problème SAT

CONACQ est donc un système visant à répondre au problème d'acquisition de réseaux de contraintes. Pour ce faire, il propose de construire le réseau en se basant sur des solutions et non-solutions du problème à résoudre. Un exemple positif pourra éliminer certaines contraintes car cet exemple ne peut être rejeté par le réseau, alors qu'un exemple négatif informe qu'il existe au moins une contrainte qui le rejette. Cela revient à maintenir un espace des versions, indiquant d'une part les contraintes ne pouvant pas être dans le réseau, raffiné grâce aux exemples positifs, et d'autre part en représentant les contraintes encore possibles, construites à partir des exemples négatifs.

Mais un des atouts principaux de CONACQ est la possibilité d'encoder le problème grâce à une formulation SAT à la fois efficace et intuitive. Ainsi dans [3, 2], les auteurs montrent qu'en représentant les deux bornes par une seule théorie clausale, on peut à la fois représenter le réseau de contraintes et rendre le traitement des exemples simple à exprimer. Nous allons détailler cette formalisation dans la suite et nous nous en servirons dans les sections suivantes pour décrire notre approche.

Nous utilisons les notations suivantes. Nous représentons un CSP par un triplet $\langle X, D, C\rangle$, où X est l'ensemble des variables, D les domaines et enfin C les contraintes. Une contrainte est représentée par $\langle S, R\rangle$ où S est la portée de la contrainte et R sa relation de satisfaction. Cet ensemble C est dans le cas de CONACQ inconnu et doit être construit grâce aux exemples fournis par l'utilisateur et une bibliothèque de contraintes, que nous notons C_X, représentant toutes les contraintes autorisées sur les variables X. Il s'agit alors de trouver un sous-ensemble C de C_X, tel que les exemples positifs soient solutions de C et les exemples négatifs non-solutions.

Dans CONACQ, la présence d'une contrainte de C_X est représentée par une variable binaire qui, si elle est vraie, indique que la contrainte appartient au réseau et, si elle est fausse, qu'elle n'y est pas. Il faut être vigilant sur le fait qu'un booléen à faux représente l'absence de la contrainte et non sa négation. Ainsi, pour le CSP où $C = \emptyset$, toute affectation des variables est solution et les booléens associés aux contraintes sont à faux. Afin de simplifier la notation, nous ne considérons que des contraintes binaires, cependant la formalisation peut être étendue aux contraintes n-aires. Pour chaque contrainte $c(X_i, X_j)$ de C_X, nous associons donc une variable binaire que nous noterons b^c_{ij}. Nous utiliserons également une notation simplifiée pour $c(X_i, X_j)$ dans les exemples qui est c_{ij} comme utilisées dans [3, 2].

Exemple

Considérons l'ensemble des variables $\{X_0, X_1, X_2, X_3\}$ *ayant toutes comme domaine*

{0,1,2,3}. Nous utiliserons comme types de contraintes $\{\leq, \neq, \geq\}$. Nous supposons que le problème cible est $X_0 \leq X_3$, $X_1 \neq X_2$ et $X_2 \geq X_3$. Les variables binaires associées à ces contraintes, $\leq_{03}$, $\neq_{12}$ et $\geq_{23}$, devront être mise à vrai à la fin de l'acquisition du réseau alors que les autres seront à faux.

Les exemples nécessaires à CONACQ sont des solutions et non-solutions du problème cible. Autrement dit, ce sont des affectations pour chacune des variables du problème d'une valeur de son domaine. Une substitution peut se représenter comme une fonction qui pour chaque variable retourne la valeur qui lui est attribuée, $\sigma = (X_0 = v_0, X_1 = v_1, \ldots, X_n = v_n)$ où $X = \{X_i \mid 0 \leq i \leq n\}$ et $v_i \in D_i$ pour tout i de 0 à n. Dans la suite, nous simplifierons cette notation en indiquant uniquement les valeurs : $(v_0, v_1, \ldots, v_n)$.

Exemple (suite)

Dans l'exemple précédent, $e_1 = (0,0,0,0)$ représente un exemple négatif ne satisfaisant pas la contrainte $X_1 \neq X_2$ alors que $e_2 = (0,1,0,0)$ est un exemple positif.

Maintenant que les exemples sont représentés, il nous reste à présenter comment un exemple apporte une connaissance sur les valeurs des booléens associés aux contraintes. Selon que l'exemple soit positif ou négatif, nous avons dit que le traitement serait différent. Avant de présenter ces traitements, il est important de définir un ensemble utilisé également dans la suite, il s'agit de l'ensemble des contraintes non satisfaites. Nous rappelons qu'étant donnée une substitution σ des variables d'un CSP, une contrainte $c(X_i, X_j)$ est satisfaite si $(\sigma(X_i), \sigma(X_j))$ appartient à la relation de satisfaction de la contrainte.

Définition 2.1 ($\mathcal{K}(e)$)**.** *Étant donné un exemple e, on définit l'ensemble $\mathcal{K}(e)$ comme l'ensemble des variables booléennes b^c_{ij} telles que la contrainte $c(X_i, X_j)$ ne soit pas satisfaite dans e.*

Plus formellement,

$$\mathcal{K}(e) = \{b^c_{ij} \mid c(X_i, X_j) \in C_X \wedge c(X_i, X_j) = \langle S, R\rangle \wedge (e(X_i), e(X_j)) \notin R\}$$

Pour chaque exemple, CONACQ va exploiter cet ensemble pour apporter une information complémentaire. L'idée est que CONACQ part d'une théorie clausale K vide au départ de l'algorithme. Il ajoutera dans cette théorie des clauses sur les variables b^c_{ij}. L'apprentissage sera terminé quand il n'y aura plus qu'une seule solution au problème SAT correspondant à K. Plus précisément, CONACQ exploitera l'ensemble $\mathcal{K}(e)_{[K]}$ restreint aux variables booléennes non encore fixées dans K.

Exemple (suite)

Pour les exemples e_1 et e_2, on a :

$\mathcal{K}(e_1)_{[K]} = \{\neq_{01}, \neq_{02}, \neq_{03}, \neq_{12}, \neq_{13}, \neq_{23}\}$

et

$\mathcal{K}(e_2)_{[K]} = \{\geq_{01}, \neq_{02}, \neq_{03}, \leq_{12}, \leq_{13}, \neq_{23}\}$

CONACQ traite les exemples les uns après les autres en ajoutant des clauses selon que l'exemple est une solution ou une non-solution.

Dans le cas où l'exemple est une solution, on peut déduire que les contraintes représentées dans $\mathcal{K}(e)_{[K]}$ ne peuvent être présentes dans le réseau. En effet, si une de ces contraintes étaient présentes dans le CSP, alors par définition, cet exemple ne pourrait être une solution. On ajoutera donc pour chaque b^c_{ij} de $\mathcal{K}(e)_{[K]}$, $\neg b^c_{ij}$ à K.

Quand l'exemple est une non-solution, alors il existe une contrainte représentée dans $\mathcal{K}(e)_{[K]}$ qui appartient au réseau. En effet, si l'ensemble C_X est suffisant pour décrire le problème, au moins une des contraintes non satisfaites doit être présente pour rejeter l'exemple. On ajoutera donc la clause $\bigvee_{b^c_{ij} \in \mathcal{K}(e)_{[K]}} b^c_{ij}$ à la théorie clausale K.

Exemple (suite)

Ainsi, si CONACQ reçoit les exemples e_1 et e_2 dans cet ordre, on obtient progressivement les théorie K ci-dessous.

e	$\mathcal{K}(e)_{[K]}$	l	K
$e_1 =$ $(0,0,0,0)$	$\{\neq_{01}, \neq_{02}, \neq_{03},$ $\neq_{12}, \neq_{13}, \neq_{23}\}$	$-$	$(\neq_{01} \vee \neq_{02} \vee \neq_{03}$ $\vee \neq_{12} \vee \neq_{13} \vee \neq_{23})$
$e_2 =$ $(0,1,0,0)$	$\{\geq_{01}, \neq_{02}, \neq_{03},$ $\leq_{12}, \leq_{13}, \neq_{23}\}$	$+$	$(\neq_{01} \vee \neq_{12} \vee \neq_{13})$ $\wedge\neg \geq_{01} \wedge\neg \neq_{02}$ $\wedge\neg \neq_{03} \wedge\neg \leq_{12}$ $\wedge\neg \leq_{13} \wedge\neg \neq_{23}$

À remarquer que e_2 permet de fixer des valeurs pour plusieurs variables booléennes et par conséquent simplifie la première clause apprise avec e_1.

CONACQ va ainsi traiter tous les exemples fournis par l'utilisateur. Il pourra cependant s'arrêter si toutes les variables booléennes sont fixées. En effet, dans ce cas il n'y a plus rien à apprendre et l'espace des versions a convergé sur une unique hypothèse : le réseau cible.

Ceci est un point très intéressant de CONACQ puisqu'il permet de garantir à l'utilisateur que l'algorithme d'acquisition est complètement terminé et que le réseau retourné est exactement celui ciblé par l'utilisateur. CONACQ propose différents mécanismes, non

détaillés dans ce papier, permettant d'améliorer son efficacité comme les règles de redondance, les ensembles de conflit ou encore la mise à jour d'un *backbone*.

2.2 Générateurs de requêtes

CONACQ a évolué pour répondre aux limites qui lui était reprochées vers la résolution d'un problème d'acquisition de réseaux de contraintes avec requêtes (voir [4] pour la version dirigée par les requêtes). Au lieu de demander à l'utilisateur de fournir les exemples, le système doit poser des questions à l'utilisateur : des requêtes. Ce cadre d'apprentissage est assez large[1] et nous ne nous intéressons qu'aux requêtes d'appartenance, consistant, dans le cas de l'acquisition de réseaux de contraintes, en des affectations des variables du CSP.

Définition 2.2 (Requête complète)**.** *Dans le cas de la résolution d'un problème d'acquisition d'un CSP $\langle X, D, C\rangle$, où C est inconnu, on appelle* requête complète *une affectation de toutes les variables de X à une valeur de leur domaine.*

Nous disons que la requête est positive, si l'utilisateur (l'oracle) répond que l'affectation est une solution du problème. Nous parlons sinon de requête négative.

Il faut remarquer que nous définissons ici une requête *complète* alors que dans [4] il est question simplement de requêtes. Nous avons ajouté cette qualification de « complète » pour faire la différence dans la suite avec les *requêtes partielles* que nous introduirons.

Les requêtes complètes sont donc des exemples générés par le système et étiquetés par l'utilisateur. L'intérêt est que l'utilisateur n'a plus à chercher par lui même ces exemples ce qui rend ainsi son travail plus simple. Les exemples précédemment décrits sont remplacés par ces requêtes dans le processus d'apprentissage de CONACQ. Le traitement des exemples reste le même et seule la génération de requêtes est à ajouter au système.

Pour générer des requêtes, les auteurs proposent différentes approches : affectation aléatoire des variables, génération d'une requête avec un « petit » $\mathcal{K}(e)_{[K]}$ ou encore une requête avec un « gros » $\mathcal{K}(e)_{[K]}$. Ils font également remarquer qu'en générant les requêtes de cette manière, il se peut que certaines requêtes ne soient pas intéressantes. En effet, il peut arriver qu'une requête ne contienne pas de nouvelle information car déjà rejetée de l'espace des versions. On parlera alors de requête redondante.

Définition 2.3 (Requête redondante)**.** *Une requête redondante est une requête qui ne permet pas d'apprendre de nouvelles informations. Étant donnée une requête e, il y a deux cas de requêtes redondantes :*

- *soit $\mathcal{K}(e)_{[K]} = \emptyset$ et dans ce cas on peut automatiquement détecter que e est une solution ;*
- *soit $\mathcal{K}(e)_{[K]}$ est un sur-ensemble des littéraux d'une clause présente dans K et donc e est une non-solution.*

Ainsi ces requêtes ne présentent aucun intérêt et il n'est pas nécessaire de demander à l'utilisateur de les étiqueter.

Le dernier point à évoquer dans cette section concerne un biais utilisé dans l'apprentissage par requêtes consistant à commencer l'apprentissage en fournissant au système d'apprentissage, un exemple positif (voir [1]). D'un point de vue cognitif, cela s'explique par le fait que l'enseignant doit déjà montrer un exemple positif de ce qu'il veut faire apprendre : « je veux que tu généralises les choses ressemblant à cela ». Le système d'apprentissage commence alors à poser des questions en essayant d'apprendre le concept se cachant derrière ce premier exemple. C'est également le cas dans CONACQ où les expériences menées dans [4] commencent avec une solution du problème cible. Cela pose à nouveau des difficultés car cette solution doit être fournie par l'utilisateur. Or dans notre approche, c'est justement cette solution que l'utilisateur cherche à trouver. Nous verrons cependant que ce cas n'est pas nécessaire pour la *recherche interactive* décrite dans la prochaine section.

3 Extension de CONACQ aux requêtes partielles

Dans le cadre de la recherche interactive, les requêtes peuvent prendre la forme d'affectation partielle des variables du CSP cible. Cela présente plusieurs intérêts : les requêtes étant plus petites, elles sont plus faciles à étiqueter pour l'utilisateur, il est plus facile de produire des requêtes qui correspondent à des solutions partielles pour les problèmes ayant peu de solutions, ou encore elles peuvent être plus simples à générer dans le cas par exemple de la simplification de clauses. De plus dans le cas de notre approche basée sur l'arbre de résolution d'un CSP, ces requêtes sont très naturelles à générer puisqu'il s'agit pour l'utilisateur d'étiqueter les nœuds de l'arbre de recherche que le système ne sera pas capable d'étiqueter seul (dans le cas des requêtes redondantes, le système peut décider seul).

Étant donné un CSP $\langle X, D, C\rangle$, on parle d'affectation partielle des variables X pour une substitution incomplète des variables de X à une valeur de leur domaine. Nous définissons la sémantique des requêtes

ainsi : une solution partielle est une affectation partielle où les contraintes concernées uniquement par les variables affectées de X sont satisfaites.

Une requête partielle est donc une question posée à l'utilisateur sous la forme d'une affectation partielle. Si c'est une solution partielle, elle est étiquetée positivement, sinon négativement. Pour les variables non-substituées dans une requête partielle nous utiliserons la notation Ω signifiant que la valeur pour cette variable est inconnue. Par exemple, en considérant un CSP avec trois variables X_1, X_2 et X_3 partageant le même domaine $\{1, 2\}$, une requête complète pourrait être $(1, 2, 1)$ et une requête partielle où X_2 n'est pas substituée, pourrait être $(1, \Omega, 1)$.

Pour finir avec les notations, nous introduisons une notion intéressante appelée la partie pertinente d'une requête.

Définition 3.1 (Partie pertinente d'une requête)**.** *Étant donnée une requête $e = (a_1, a_2, \ldots, a_n)$, où les a_i sont des constantes ou le symbole Ω, nous disons que $a_i \in e$ est pertinente si $a_i \neq \Omega$ et qu'il existe $b^c_{ij} \in \mathcal{K}(e)_{[K]}$ ou $b^c_{ji} \in \mathcal{K}(e)_{[K]}$.*

La partie pertinente $r(e)$ d'une requête e est le sous-ensemble minimal de e contenant les constantes pertinentes.

L'intérêt d'une telle notion est que la partie pertinente est la seule chose utile à montrer à l'utilisateur et donc nous pouvons réduire la taille de la requête de cette manière. Par la suite, nous ne considérons que les parties pertinentes quand nous parlerons de la taille d'une requête.

Pour éviter une énumération de toutes les affectations possibles, notre approche consiste à apprendre progressivement les contraintes du problème cible. La recherche se fera alors d'une manière habituelle en programmation par contraintes, alternant les phases de branchement, affectant une valeur à une variable de X, et les phases de propagation, supprimant des domaines les valeurs ne pouvant faire partie d'une solution. Après chaque étape de propagation, notre algorithme, décrit dans la section 4, évalue si l'affectation courante est une solution (partielle). S'il n'est pas capable de le déterminer seul, il soumet une requête à l'utilisateur et, dépendant de la réponse, ajoute des contraintes au CSP. La redondance d'une requête partielle est identique à celle d'une requête complète. On peut donc déterminer automatiquement si une requête redondante est une solution partielle ($\mathcal{K}(e)_{[K]}$ vide) ou s'il s'agit d'une non-solution ($\mathcal{K}(e)_{[K]}$ est un sur-ensemble d'une clause de K).

Les traitements de CONACQ pour les requêtes complètes dépendent uniquement de l'ensemble $\mathcal{K}(e)_{[K]}$. Pour étendre ces traitements aux requêtes partielles, il suffit de redéfinir cet ensemble afin qu'il prenne en compte le fait que certaines variables ne sont pas substituées. Plus précisément, il faut redéfinir $\mathcal{K}(e)$ de la manière suivante :

$$\begin{aligned}\mathcal{K}(e) = \{b^c_{ij} \mid \quad & c(X_i, X_j) \in C_X \\ & \wedge\ c(X_i, X_j) = \langle S, R\rangle \\ & \wedge\ e(X_i) \neq \Omega\ \wedge\ e(X_j) \neq \Omega \\ & \wedge\ (e(X_i), e(X_j)) \notin R\}\end{aligned}$$

Alors qu'une requête partielle est plus facile à étiqueter pour l'utilisateur, elle contient en contre-partie potentiellement moins d'information. Or, les auteurs de CONACQ évaluent leur approche en regardant le nombre de requêtes nécessaires pour faire converger leur système. Cette manière d'évaluer ne prenant pas en compte la taille des requêtes, nous proposons une mesure correspondant à $\#q \times |q|$, un genre d'aire, de surface, qui sera noté $m(q)$ dans la suite. Elle permet de relativiser le nombre de requêtes avec leur taille.

4 Génération de requêtes biaisée par la résolution du CSP

Notre algorithme est une recherche usuelle alternant la propagation des contraintes et les branchements. Pour éviter une exploration complète, nous apprenons les contraintes du problèmes grâce à des requêtes. Pour chaque nœud de l'arbre de recherche, il existe plusieurs cas. Nous rappelons qu'un nœud représente une affectation partielle des variables du CSP. Si l'affectation (partielle) représentée par le nœud est une requête redondante, nous pouvons décider sans l'utilisateur si l'exploration peut continuer dans cette branche dans le cas où nous avons une solution partielle ou bien s'arrêter et continuer l'exploration dans une nouvelle branche. Dans le cas où la requête n'est pas redondante, il faut demander à l'utilisateur de l'étiqueter pour continuer la recherche. Nous commençons donc par résoudre un problème sans ses contraintes et ajoutons les contraintes apprises lors de l'apprentissage. Cette démarche est similaire à celle que l'on peut trouver dans les algorithmes de recherche *branch-and-bound*. Dans ces derniers, à chaque solution trouvée dans l'arbre de recherche une contrainte est ajouté pour forcer la prochaine solution à être meilleure. Dans notre cas, à chaque requête nous ajoutons une contrainte pour éviter de parcourir des états de recherche similaires (requêtes redondantes). Pour mettre en avant cette analogie, nous appelons notre approche *branch-and-learn*. Ce n'est évidemment pas la seule façon d'exploiter les requêtes partielles mais celle-ci a

l'avantage de fournir un cadre clair à l'utilisateur. La machine cherche à résoudre le problème dès le départ et interagit avec l'utilisateur pour cerner le modèle jusqu'à trouver une solution.

Pour généraliser, nous voyons notre processus comme un générateur de requête pour un système comme CONACQ qui serait capable de gérer les requêtes partielles. Dans [4], les auteurs proposent de générer les requêtes à la volée en fonction de l'état du processus d'apprentissage. Leur générateur ne produit que des requêtes non redondantes, utiles pour améliorer la théorie clausale courante. Les différentes étapes d'un tel système sont :

- Génération d'une requête q ;
- Étiquetage de q par l'utilisateur ;
- Mise à jour de K avec q en fonction de la réponse.

Nous présentons notre algorithme de résolution de ce point de vue. Notre approche consiste donc en deux modules, un premier recherchant une solution dans l'arbre de résolution et qui retourne une requête quand cela est nécessaire, et un autre mettant à jour le modèle de contraintes du problème à la manière de CONACQ mais gérant les requêtes partielles.

En utilisant comme biais de génération de requêtes l'arbre de résolution du CSP, nous commençons par la racine de l'arbre décrivant une affectation partielle vide (aucune variable n'a de substitution). Ensuite, un nœud n_2 est obtenu à la suite un nœud n_1 si l'affectation partielle représentée par n_2 est un sur-ensemble de celle de n_1. La structure de l'arbre dépend en fait de la manière dont sont produits les nouveaux nœuds à partir d'un existant. Par exemple, une stratégie usuelle consiste à fixer une valeur pour une variable. Il y a alors deux nouveaux nœuds : celui où la variable est affectée la valeur et celui où cette valeur est retirée du domaine de la variable. Mais d'autres stratégies existent comme par exemple fixer plus d'une variable. Ces stratégies forment les stratégies de branchement et pour généraliser nous dirons qu'une stratégie ajoute seulement des contraintes aux nouveaux nœuds. Ainsi, pour fixer une variable X_i à la valeur a, on ajoutera la contrainte $X_i = a$ pour créer le nœud. L'autre nœud où l'on retire a du domaine de X_i aura la contrainte $X_i \neq a$.

Comme le CSP est progressivement appris par le module de type CONACQ, il est important, avant de traiter un nœud, de mettre à jour ses contraintes. En effet, nous partons d'un nœud où il n'y a aucune contrainte et il faut les ajouter dès que celles-ci sont connues. Pour se faire nous proposons un encodage proche de celui proposé par CONACQ. Le CSP de départ de notre arbre est composé des variables X associées à leur domaine D. Nous ajoutons également les variables booléennes b^c_{ij} comme défini dans CONACQ. Pour résumer, les variables du CSP sont les suivantes :

$$\forall X_i \in X, X_i \in D_{X_i}$$
$$\forall c(X_i, X_j) \in C_X, b^c_{ij} \in \{0, 1\}$$

Les contraintes de départ de ce CSP sont alors les suivantes :

$$\forall c(X_i, X_j) \in C_X, b^c_{ij} \Rightarrow c(X_i, X_j)$$

Ainsi, une affectation d'un booléen décidera si une contrainte sera « activée » ou pas. Ces variables booléennes ayant la même sémantique que pour CONACQ, si l'une d'elles est mise à faux alors la contrainte n'est pas présente (ce qui ne signifie pas que sa négation le soit). Cela explique pourquoi nous avons une implication plutôt qu'une équivalence.

La mise à jour d'un nœud revient à fixer à vrai ou faux les variables booléennes dont la valeur est connue dans le module d'apprentissage des contraintes. On ajoutera également les clauses apprises afin de limiter les requêtes redondantes.

Maintenant que les nœuds sont définis, le dernier point concerne la manière dont le générateur de requêtes va les explorer. Plusieurs stratégies peuvent être utilisées comme la recherche en profondeur (DFS), le *restart*, le *branch-and-bound*... Nous nommons ces stratégies les stratégies d'exploration.

L'algorithme de la Figure 1 résume le processus de génération de requêtes biaisé par l'arbre de résolution. Premièrement, introduisons quelques notations. Soit *nodes* l'ensemble ordonné des nœuds qui attendent d'être traités. Son implémentation dépend de la stratégie d'exploration. L'opération *next* de *nodes* retourne le premier nœud de l'ensemble et *erase* supprime le nœud en argument de l'ensemble. Pour un nœud n, l'opération *update* met à jour le modèle de contraintes du nœud en fonction de la théorie clausale courante K, *propagate* réalise la propagation des contraintes et *query* extrait la requête représentée par le nœud. Finalement, l'opération *branching*, dépendant de la stratégie de branchement, produit tous les fils possibles du nœud donné en paramètre.

L'algorithme commence par récupérer le prochain nœud. Après la mise à jour des contraintes et leur propagation, la requête q est extraite du nœud. Si la requête n'est pas redondante, il faut la soumettre à l'utilisateur et l'algorithme s'arrête en la retournant. Sinon, nous pouvons déterminer s'il s'agit d'une solution (partielle) ou non. L'algorithme produit alors de nouveaux nœuds si la requête n'est pas complète et procède à un appel récursif pour explorer les nœuds

Algorithm : GENERATE

```
1.  cur ← nodes.next()
2.  cur.update(K)
3.  cur.propagate()
4.  q ← cur.query()
5.  if q is non-redundant then
6.    return q
7.  else
8.    nodes.erase(cur)
9.    if q is a (partial) solution then
10.     if q is a complete query then
11.       cur is a solution
12.     else
13.       nodes ← nodes ∪ branching(cur)
14.     end if
15.   end if
16.   GENERATE
17. end if
```

FIGURE 1 – Génération de requêtes biaisées par l'arbre de résolution

suivants. Reste le cas particulier où il y a une solution (requête complète redondante positive). Dans le cas où nous cherchons justement une solution le processus s'arrêtera en fournissant cette solution à l'utilisateur.

Exemple

Pour illustrer notre algorithme, nous l'appliquons au problème suivant. Le problème a quatre variables $\{X_0, X_1, X_2, X_3\}$ *avec comme domaine* $\{0, 1, 2, 3\}$ *et les types de contraintes sont* $\{\leq, \neq, \geq\}$. *Les contraintes du problème cible sont* $X_0 \leq X_3$, $X_1 \neq X_2$ *et* $X_2 \geq X_3$.

Nous utilisons une recherche en profondeur d'abord, en branchant sur les variables dans l'ordre suivant X_0, X_1, X_2 *et* X_3 *avec la plus petite valeur possible. La Figure 2 représente l'arbre de recherche exploré par notre algorithme. Les nœuds bleus (*e_1 *et* e_2*) représentent des requêtes redondantes et ne sont pas soumis à l'utilisateur. Les nœuds verts (*e_3, e_5 *et* e_6*) et rouges (*e_4*) représentent respectivement des requêtes positives et négatives étiquetées par l'utilisateur.*

Le tableau de la Figure 3 montre les contraintes apprises avec les requêtes successives. La première colonne donne les requêtes soumises à l'utilisateur. La seconde montre les $\mathcal{K}(e)_{[K]}$ *correspondant aux requêtes et la troisième la réponse de l'utilisateur,* $+$ *signifiant positif et* $-$ *négatif. La dernière montre les contraintes ajoutées à* K *qui seront également ajoutées au CSP en construction lors de la mise à jour.*

La recherche commence avec deux requêtes redondantes où le $\mathcal{K}(e)_{[K]}$ *est vide. Comme ces requêtes correspondent à des solutions (partielles) la recherche continue. La première requête soumise est* e_3 *qui est une solution partielle permettant d'éliminer la contrainte* $X_0 \neq X_1$. *Pour la seconde, l'utilisateur répond négativement et notre système apprend la clause* $\neq_{02} \vee \neq_{12}$. *La recherche retourne alors en arrière et branche sur* $X_2 = 1$. *Ensuite, nous avons deux requêtes positives pour obtenir une solution au problème.*

Il est clair que sur un exemple jouet comme celui-ci, il est difficile de voir l'intérêt apporté par l'apprentissage de contraintes, qui est censé éviter de parcourir exhaustivement tout l'espace de recherche. Nous verrons cependant dans la section suivante qu'en continuant la recherche sur ce problème les contraintes apprises permettent d'élaguer une bonne partie de l'arbre de recherche. Sur des problèmes vraiment complexes, où peu de solutions existent, les contraintes apprises permettent également de converger plus efficacement vers une solution.

L'utilisateur peut souhaiter plusieurs solutions, si la première ne lui convenait pas tout à fait (même si elle respecte les contraintes de son problème). Dans ce cas, nous pouvons continuer la recherche là où elle avait été arrêtée et rechercher la prochaine solution, voire toutes les solutions. Pour cela, nous poursuivons l'exploration de l'arbre de recherche de la même manière que précédemment.

En faisant ainsi, nous pouvons nous demander s'il est possible d'apprendre le modèle exact du problème. Comme nous le verrons dans la partie concernant les expériences que nous avons menées, notre méthode ne permet pas de garantir une convergence de l'espace des versions comme c'était le cas dans CONACQ. Cela est dû à la manière dont nous générons nos requêtes et à l'absence de règles de redondances dans notre système. Cependant les contraintes apprises auxquelles on ajoute les clauses sur les variables non fixées sont suffisantes pour reproduire la résolution du CSP sans aucune requête pour l'utilisateur. Nous finissons cette section en montrant que les ensembles de solutions du CSP cible et celles du CSP construit à partir des contraintes et clauses apprises pendant la résolution sont les mêmes. Avant cela nous introduisons quelques notations nécessaires. Dans la suite, nous noterons le CSP que cible l'utilisateur CSP_c. L'ensemble des solutions d'un CSP nommé P sera noté $sol(P)$. Enfin, étant donnés le $\text{CSP}_c = \langle X, D, C \rangle$, la bibliothèque de contraintes C_X et une théorie clausale K, nous définissons le CSP CSP_K de la manière suivante :

- $\text{CSP}_K = \langle X', D', C' \rangle$
- $X' = X \cup \{b^c_{ij} \mid c(X_i, X_j) \in C_X\}$
- $D' = D \cup \{D_{b^c_{ij}} = \{0, 1\} \mid c(X_i, X_j) \in C_X\}$
- $\forall c(X_i, X_j) \in C_X, (b^c_{ij} \Rightarrow c(X_i, X_j)) \in C'$
- Pour toute clause m de K, $m \in C'$

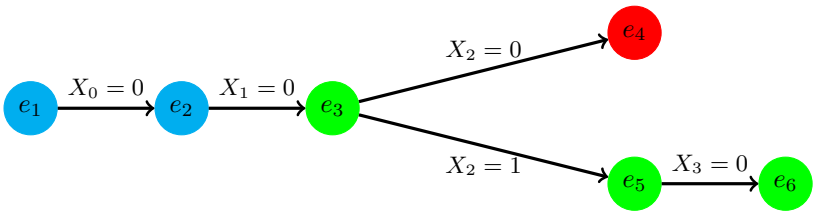


FIGURE 2 – Recherche d'une solution

e	$\mathcal{K}(e)_{[K]}$	Label	Added constraints
$e_3 = (0, 0, \Omega, \Omega)$	$\{\neq_{01}\}$	+	$\neg \neq_{01}$
$e_4 = (0, 0, 0, \Omega)$	$\{\neq_{02}, \neq_{12}\}$	−	$\neq_{02} \vee \neq_{12}$
$e_5 = (0, 0, 1, \Omega)$	$\{\geq_{02}, \geq_{12}\}$	+	$\neg \geq_{02} \wedge \neg \geq_{12}$
$e_6 = (0, 0, 1, 0)$	$\{\neq_{03}, \geq_{23}, \neq_{13}\}$	+	$\neg \neq_{03} \wedge \neg \geq_{23} \wedge \neg \neq_{13}$

FIGURE 3 – Contraintes apprises pendant la recherche interactive d'une solution

Ce CSP possède les mêmes variables et domaines que le CSP cible, les mêmes contraintes réifiées permettant de faire le lien entre les variables booléennes et les contraintes de C_X ainsi que les clauses de la théorie clausale sous forme de contraintes réifiées quand les clauses contiennent plus d'un littéral.

Théorème 4.1. *Étant donnés un CSP cible* CSP_c, C_X *permettant de le représenter et la théorie clausale* K *obtenue en cherchant toutes les solutions avec un parcours complet de l'arbre de recherche, nous avons* $sol(\text{CSP}_c) = sol(\text{CSP}_K)$.

Démonstration Pour obtenir K, notre méthode consiste de partir d'une théorie vide K_0 et d'ajouter à chaque requête, soit une clause m si la requête est négative, soit un ensemble de clauses unitaires de la forme $\neg b_{ij}^c$ si elle est positive.

Nous avons donc une séquence de raffinements $K_0, K_1, \ldots, K_n = K$ vérifiant l'implication logique $K_{i+1} \rightarrow K_i$.

Montrons que $sol(\text{CSP}_c) \subseteq sol(\text{CSP}_K)$. Supposons qu'il existe une requête complète e telle que $e \in sol(\text{CSP}_c)$ et $e \notin sol(\text{CSP}_K)$. Il existe donc un K_i rejettant e ce qui est une contradiction avec le parcours complet de l'arbre de recherche. Montrons que $sol(\text{CSP}_K) \subseteq sol(\text{CSP}_c)$. Trivialement, nous avons $sol(\text{CSP}_c) \subseteq sol(\text{CSP}_{K_0})$. Supposons maintenant qu'il existe i compris entre 0 et n tel que $\forall s \in sol(\text{CSP}_c), s \in sol(\text{CSP}_{K_i})$. Nous montrons maintenant que $\forall s \in sol(\text{CSP}_c), s \in sol(\text{CSP}_{K_{i+1}})$. Dans le cas d'une requête positive en passant de K_i à K_{i+1}, les solutions de $\text{CSP}_{K_{i+1}}$ restent les mêmes que celles de CSP_{K_i}, la différence entre les deux théories résidant uniquement dans l'élimination de contraintes (ajout de clauses unitaires du type $\neg b_{ij}^c$). On a donc $\forall s \in sol(\text{CSP}_c), s \in sol(\text{CSP}_{K_{i+1}})$. Dans le cas d'une requête négative, nous avons $K_{i+1} = K_i \cup \{m\}$ où m est une clause. Alors, $sol(\text{CSP}_{K_{i+1}}) = sol(\text{CSP}_{K_i}) \cap (\bigcup_{b_{ij}^c \in m} sol(c(X_i, X_j))$, où $sol(c(X_i, X_j))$ est l'ensemble des affectations de X satisfaisant la contrainte $c(X_i, X_j)$. Par hypothèse $sol(\text{CSP}_c) \subseteq sol(\text{CSP}_{K_i})$, il nous reste donc à montrer $sol(\text{CSP}_c) \subseteq (\bigcup_{b_{ij}^c \in m} sol(c(X_i, X_j))$. Supposons qu'il existe une solution s de $sol(\text{CSP}_c)$ telle que pour tout b_{ij}^c de m, s n'est pas dans $sol(c(X_i, X_j))$. Or d'après la définition des traitements des requêtes, les b_{ij}^c de m devraient être à 0 comme $s \in sol(\text{CSP}_c)$. La requête aurait donc dû être positive ce qui mène à une contradiction.

Nous obtenons donc bien l'égalité entre ces deux ensembles de solutions.

5 Évaluation de la recherche interactive

Nous proposons deux heuristiques directement inspirées de la version dirigée par les requêtes de CONACQ. Nommées respectivement `MaxK` et `MinK`, ces heuristiques choisissent de compléter une instance partielle des variables en affectant une valeur à une variable telle que la nouvelle instance e créée possède respectivement le plus grand $\mathcal{K}(e)_{[K]}$ possible ou le plus petit. Afin de pouvoir comparer ces heuristiques, nous utiliserons également `Dom` qui affecte à la variable ayant le plus petit domaine la plus petite valeur de son domaine.

Pour étudier le comportement de notre approche, nous avons utilisé des problèmes ayant peu de solutions comme le problème du zèbre. Les spécifications de ces problèmes sont données en annexes A de [11]. Notre motivation initiale étant d'aider l'utilisateur à trouver des solutions à des problèmes qu'il ne peut résoudre facilement nous avons donc préféré ce genre de problème.

La plupart n'ayant qu'une solution, une comparaison entre notre approche et CONACQ n'a pas réellement de sens. De plus, CONACQ ne gére pas les requêtes partielles et son objectif diffère (trouver une ou toutes les solutions pour nous, trouver les contraintes pour CONACQ). On peut cependant faire remarquer que sur les jeux de données testés dans [4] notre approche nécessite plus de requêtes. Cela est à relativiser car la taille de nos requêtes est naturellement plus petite que celles de CONACQ qui n'utilise pas de requêtes partielles. Nous proposons de concentrer notre analyse sur notre approche.

Nous avons effectué deux séries d'expériences, celles-ci diffèrent par leur utilisation de la simplification de clauses, mécanisme créé dans CONACQ visant à simplifier une clause apprise en générant des requêtes avec un $\mathcal{K}(e)_{[K]}$ de taille 1. À noter que cette technique a été étendue à l'utilisation de requêtes partielles permettant de générer plus facilement les requêtes. Les Figures 4 et 5 résument les résultats obtenus. Le sens des colonnes est le suivant :

- # variables : le nombre de variables du CSP
- # contraintes cibles : le nombre de contraintes du CSP quand il est écrit « à la main » ;
- stratégie : stratégie d'exploration de l'arbre de recherche ;
- #q : le nombre de requêtes soumis à l'utilisateur ;
- |q| : la taille moyenne des requêtes ;
- m(q) : $\#q \times |q|$;
- temps : le temps mis par le système (avec étiquetage automatique) ;
- nœuds explorés : le nombre de nœuds explorés dans l'arbre de recherche ;
- contraintes apprises : le nombre de contraintes dont le littéral est fixé (à vrai ou à faux).

Comme nous pouvons le voir sur la Figure 4, l'utilisation d'heuristiques telles que `MaxK` et `MinK` ne permet pas de faire baisser le nombre de requêtes par rapport à une heuristique comme `Dom`. Cela peut s'expliquer par le fait que ces heuristiques ont été pensées pour chercher des requêtes visant à apprendre le réseau de contraintes plutôt qu'à arriver rapidement vers une solution. On peut s'interroger sur le fait que `MaxK` produit des requêtes plus petites que `MinK` alors que l'intuition aurait été l'inverse. En fait, cela s'explique par le fait qu'en cherchant à maximiser la taille du $\mathcal{K}(e)_{[K]}$, on augmente le nombre de contraintes violées par la requête et donc la probabilité que l'une d'elles appartienne au réseau cible. Par conséquent, on retournera plus rapidement en arrière dans l'arbre de recherche et on gardera donc des requêtes assez petites.

Dans la Figure 5, nous nous sommes intéressés à l'utilisation de la simplification de clauses. Comme dit précédemment, `MaxK` et `MinK` ont été conçus dans l'esprit de trouver les contraintes du réseau cible et si l'on se contente des clauses (rarement unaires) apprises quand on a une requête négative, ces clauses n'ont que peu d'incidence sur l'exploration de l'arbre de recherche. Par contre, en recherchant dans ces clauses, une contrainte à ajouter à notre réseau, nous pouvons constater que chercher à simplifier les clauses produira plus de requêtes mais également de plus petites requêtes. Si, nous nous intéressons à la colonne m(q), nous pouvons voir qu'une heuristique comme $MaxK$ devient très intéressante.

6 Conclusion

Dans ce papier, nous avons voulu fournir un nouveau regard sur les problématiques d'acquisition de réseaux de contraintes. Plutôt que de se fixer comme objectif d'apprendre le réseau, nous partons de l'hypothèse que seule une solution au problème intéresse l'utilisateur. Nous avons répondu à cela en proposant la recherche interactive. En partant d'un CSP sans aucune contrainte, nous résolvons ce problème en posant à certains moments des requêtes à l'utilisateur permettant ainsi de connaître certaines contraintes. Nous avons étendu le système CONACQ aux requêtes partielles. Ces dernières offrent des perspectives plus larges d'utilisation de ce système.

Les requêtes partielles n'ont d'ailleurs pas encore été complètement exploitées. Premièrement, leur utilisation pose des problèmes avec les règles de redondance utilisées dans CONACQ. Il serait intéressant de voir comment permettre leur utilisation afin de bénéficier de plus d'information sémantique sur les contraintes. Deuxièmement, les requêtes partielles rendent plus facile la simplification des clauses et pourraient permettre d'autres méthodes que celles qui avaient été proposées dans CONACQ.

Références

[1] Dana Angluin. Queries and concept learning. *Mach. Learn.*, 2 :319–342, April 1988.

[2] Christian Bessière, Remi Coletta, Frédéric Koriche, and Barry O'Sullivan. A sat-based version space algorithm for acquiring constraint satisfaction problems. In *ECML*, pages 23–34, 2005.

[3] Christian Bessière, Remi Coletta, Frédéric Koriche, and Barry O'Sullivan. Acquiring constraint networks using a sat-based version space algorithm. In *AAAI*. AAAI Press, 2006.

[4] Christian Bessière, Remi Coletta, Barry O'Sullivan, and Mathias Paulin. Query-driven constraint

Jeux de données	# variables	# contraintes cibles	stratégie	#q	|q|	m(q)	temps	nœuds explorés	contraintes apprises
Prudey's general store	12	45	Dom	109	5,45	594	0,61	711	121/198
			MaxK	142	4,18	594	2,06	632	171/198
			MinK	139	5,26	731	2,64	728	164/198
Allergic reactions	12	29	Dom	60	6,55	393	0,14	175	99/198
			MaxK	69	5,77	398	0,29	181	119/198
			MinK	99	6,67	660	0,20	318	143/198
Miffed millionaires	16	43	Dom	132	7,67	1012	1,11	618	237/360
			MaxK	169	5,81	982	6,07	1086	287/360
			MinK	185	6,85	1267	6,28	1178	292/360
Problème du zèbre	25	71	Dom	511	9,34	4773	92,47	3727	1741/2100
			MaxK	508	6,85	3292	1379,18	8728	2021/2100
			MinK	-	-	-	>10h	-	-

FIGURE 4 – Résultats de recherche d'une solution avec les différentes heuristiques

Jeux de données	# variables	# contraintes cibles	stratégie	#q	|q|	m(q)	temps	nœuds explorés	contraintes apprises
Prudey's general store	12	45	Dom	141	3,08	434	0,22	249	161/198
			MaxK	142	2,99	425	0,76	289	168/198
			MinK	132	3,14	414	0,49	238	162/198
Allergic reactions	12	29	Dom	101	3,08	311	0,21	155	122/198
			MaxK	98	3,04	298	0,32	163	124/198
			MinK	92	3,13	288	0,19	143	118/198
Miffed millionaires	16	43	Dom	224	3,47	777	0,47	337	288/360
			MaxK	196	3,36	659	0,76	306	277/360
			MinK	224	3,40	762	0,67	319	275/360
Problème du zèbre	25	71	Dom	589	7,38	4347	212,71	2737	1860/2100
			MaxK	384	4,68	1797	393,6	3518	2000/2100
			MinK	-	-	-	>10h	-	-

FIGURE 5 – Résultats de recherche d'une solution avec les heuristiques et de la simplification de clauses

acquisition. In Manuela M. Veloso, editor, *IJCAI*, pages 50–55, 2007.

[5] Christian Bessière, Remi Coletta, and Thierry Petit. Acquiring parameters of implied global constraints. In Peter van Beek, editor, *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 747–751. Springer, 2005.

[6] Christian Bessiere, Joël Quinqueton, and Gilles Raymond. Mining historical data to build constraint viewpoint. In Ian Miguel and Steve Prestwich, editors, *CP'06 Workshop on modelling and Reformulation*, 2006.

[7] B. M. W. Cheng, Jimmy Ho-Man Lee, and J. C. K. Wu. Speeding up constraint propagation by redundant modeling. In Eugene C. Freuder, editor, *CP*, volume 1118 of *Lecture Notes in Computer Science*, pages 91–103. Springer, 1996.

[8] Remi Coletta, Christian Bessière, Barry O'Sullivan, Eugene C. Freuder, Sarah O'Connell, and Joël Quinqueton. Semi-automatic modeling by constraint acquisition. In Francesca Rossi, editor, *CP*, volume 2833 of *Lecture Notes in Computer Science*, pages 812–816. Springer, 2003.

[9] Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *ECAI*, pages 290–295, 1988.

[10] Arnaud Lallouet, Matthieu Lopez, Lionel Martin, and Christel Vrain. On learning constraint problems. In *ICTAI (1)*, pages 45–52. IEEE Computer Society, 2010.

[11] Matthieu Lopez. *Apprentissage de problèmes de contraintes*. PhD thesis, Université d'Orléans, 2011.

[12] Jean-Francois Puget. Constraint programming next challenge : Simplicity of use. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 5–8. Springer, 2004.

[13] Barbara M. Smith. *Handbook of Constraint Programming*, chapter Modelling, pages 377–406. Elsevier, 2006.

Caractérisation de la complexité des classes de CSP définies par des motifs interdits à deux contraintes*

Martin C. Cooper **Guillaume Escamocher**

IRIT, Université de Toulouse, France
{cooper,Guillaume.Escamocher}@irit.fr

Résumé

De nouvelles classes traitables du CSP (Problème de Satisfaction de Contraintes) ont été récemment découvertes via l'étude de classes d'instances définies par l'exclusion de sous-problèmes décrits par des motifs. La caractérisation complète de toutes les classes traitables définies par des motifs interdits est un problème ambitieux. Nous démontrons une dichotomie dans le cas de motifs interdits constitués de deux contraintes. Ce travail nous a permis d'identifier de nouvelles classes traitables.

Abstract

Novel tractable classes of the binary CSP (constraint satisfaction problem) have recently been discovered by studying classes of instances defined by excluding subproblems described by patterns. The complete characterisation of all tractable classes defined by forbidden patterns is a challenging problem. We demonstrate a dichotomy in the case of forbidden patterns consisting of two constraints. This has allowed us to discover new tractable classes.

1 Introduction

Dans une instance CSP le but est de déterminer l'existence d'une affectation de valeurs à des variables telle qu'un ensemble de contraintes soient satisfaites simultanément. Une question fondamentale est l'identification de sous-problèmes traitables du CSP. Les approches classiques consistent à identifier des restrictions sur soit les types de contraintes soit l'(hyper)graphe des portées des contraintes qui impliquent l'existence d'un algorithme de complexité polynomiale. Dans certains cas, des dichotomies ont même été démontrées [2, 3, 14, 15, 16].

Récemment, une nouvelle voie de recherche s'est ouverte : l'identification de classes traitables d'instances CSP définies par l'interdiction d'un (ensemble de) sous-problème(s) spécifique(s). De nouvelles classes traitables ont été découvertes en interdisant des sous-problèmes simples sur 3 variables [9, 10]. Nous présentons dans ce papier un premier pas vers l'identification de toutes ces classes traitables, une dichotomie pour le cas particulier de sous-problèmes interdits à deux contraintes. Nous avons omis les détails de certaines preuves très longues ; les preuves complètes se trouvent dans la version arXiv de ce papier [8].

Nous définissons tout d'abord la notion de motif CSP que nous utiliserons plus tard pour décrire des classes d'instances CSP dans lesquelles le motif n'apparaît pas. Un motif peut représenter un ensemble de sous-problèmes (interdits) en laissant la compatibilité de certaines affectations indéfinie. Nous utilisons le terme *point* pour dénoter l'affectation d'une valeur à une variable. Un motif est un graphe dans lequel les sommets correspondent aux points et à la fois les sommets et les arêtes sont étiquetés. L'étiquette d'un sommet correspondant à une affectation variable-valeur $\langle v, d\rangle$ est simplement la variable v et l'étiquette d'une

*financé par le projet ANR-10-BLAN-0210.

arête entre deux sommets décrit la compatibilité de la paire d'affectations correspondante.

Définition 1. *Un* motif *est un quintuplet $\langle V, A, var, E, cpt\rangle$ comprenant : un ensemble V de variables, un ensemble A de points (affectations), une fonction de variable $var : A \to V$, un ensemble $E \subseteq \binom{A}{2}$ d'arêtes (paires non ordonnées d'éléments de A) tel que $\{a, b\} \in E \Rightarrow var(a) \neq var(b)$, et une fonction de compatibilité à valeurs booléennes $cpt : E \to \{F, T\}$. (Pour une notation plus simple, nous écrivons $cpt(a, b)$ au lieu de $cpt(\{a, b\})$).*

Pour un motif $P = \langle V, A, var, E, cpt\rangle$ et une variable $v \in V$, nous utilisons A_v pour dénoter l'ensemble d'affectations $\{a \in A \mid var(a) = v\}$ à v. Si $cpt(a, b) = T$ alors les deux affectations (points) a, b sont *compatibles* et $\{a, b\}$ est une *arête de compatibilité* ; si $cpt(a, b) = F$ alors les deux affectations a, b sont *incompatibles* et $\{a, b\}$ est une *arête d'incompatibilité*. Dans un motif, la compatibilité d'une paire de points a, b telle que $var(a) \neq var(b)$ et $(a, b) \notin E$ est *indéfinie*.

Une *instance CSP binaire* est un motif $\langle V, A, var, E, cpt\rangle$ dans lequel la compatibilité de chaque paire d'affectations à des variables distinctes est spécifiée par la fonction de compatibilité. La question correspondant à l'instance est : existe-t-il une *solution*, c'est-a-dire un ensemble d'affectations à toutes les variables dans V entièrement composé de paires compatibles ? La *contrainte* sur les variables $v_1, v_2 \in V$ est la sous instance à 2 variables $\langle\{v_1, v_2\}, A_{12}, var|_{A_{12}}, E_{12}, cpt|_{E_{12}}\rangle$ dans laquelle $A_{12} = A_{v_1} \cup A_{v_2}$ et $E_{12} = \{\{a, b\} \mid a \in A_{v_1}, b \in A_{v_2}\}$. La contrainte entre les variables v_1 et v_2 dans une instance est *non-triviale* s'il y a au moins une paire d'affectations incompatibles, c'est-à-dire $a \in A_{v_1}$ et $b \in A_{v_2}$ tels que $cpt(a, b) = F$.

Un motif est un moyen compact de représenter l'ensemble de toutes les instances obtenu en spécifiant arbitrairement la compatibilité de ses paires indéfinies. Deux motifs P et Q sont *isomorphes* s'ils sont identiques à renommage près (de variables ou d'affectations).

Définition 2. *Nous disons qu'un motif P* apparaît *dans un motif P' (et P'* contient *P) si P' est isomorphe à un motif Q dans la fermeture transitive des deux opérations suivantes (extension, collage) appliquées à P :*

extension *$P = \langle V_P, A_P, var_P, E_P, cpt_P\rangle$ est un sous-motif de $Q = \langle V_Q, A_Q, var_Q, E_Q, cpt_Q\rangle$ (et Q une extension de P) : $V_P \subseteq V_Q$, $A_P \subseteq A_Q$, $var_P = var_Q|_{A_P}$, $E_P \subseteq E_Q$, $cpt_P = cpt_Q|_{E_P}$.*

collage *l'identification (ou collage) de deux points dans $P = \langle V_P, A_P, var_P, E_P, cpt_P\rangle$ transforme P en $Q = \langle V_Q, A_Q, var_Q, E_Q, cpt_Q\rangle$: $\exists a, b \in A_P$ tels que $var_P(a) = var_P(b)$ et $\forall c \in A_P$ tel que $\{a, c\}, \{b, c\} \in E_P$, $cpt_P(a, c) = cpt_P(b, c)$. De plus, $V_P = V_Q$, $A_Q = A_P \setminus \{b\}$, $var_Q = var_P|_{A_Q}$, $E_Q = E_P \cup \{\{a, x\} \mid \{b, x\} \in E_P\}$ et $cpt_Q(a, x) = cpt_Q(b, x)$ si $\{b, x\} \in E_P$, $cpt_Q(e) = cpt_P(e)$ sinon.*

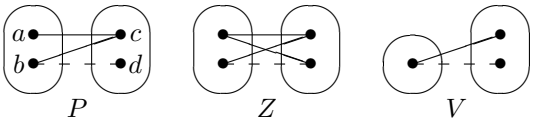


FIG. 1 – Trois motifs

Considérons les trois motifs donnés en Figure 1. Les affectations (points) sont représentes par des points, et les affectations à la même variable v sont groupés ensemble à l'intérieur d'un ovale représentant A_v. Les lignes continues représentent les arêtes de compatibilités et les lignes en pointillés représentent les arêtes d'incompatibilité. Par exemple, P comporte de 4 points $a, b \in A_{v_0}$, $c, d \in A_{v_1}$ tels que $cpt(a, c) = cpt(b, c) = T$ et $cpt(b, d) = F$. P apparaît dans Z car Z est une extension de P. P apparaît également dans V car V peut être obtenu depuis P par le collage des points a, b.

Définition 3. *Si P est un motif,* $\mathrm{CSP}(\overline{P})$ *dénote l'ensemble d'instances CSP binaires Q dans lesquelles P n'apparaît pas. Le motif P est* traitable *s'il y a un algorithme de complexité polynomiale pour résoudre* $\mathrm{CSP}(\overline{P})$ *; P est* intraitable *si* $\mathrm{CSP}(\overline{P})$ *est NP-Complet.*

2 Opérations de prétraitement

Cette section décrit certaines opérations de simplification de complexité polynomiale qui peuvent s'appliquer à une instance CSP $\langle V, A, var, E, cpt\rangle$. Si pour une variable v, A_v est un singleton $\{a\}$, alors l'*élimination d'une variable à une seule valeur* correspond à faire l'affectation a et consiste à éliminer v de V et éliminer a de A avec toutes les affectations b qui sont incompatibles avec a.

La *cohérence d'arc* [1] consiste à éliminer de A toutes les affectations a pour lesquels il y a une

variable $v \neq var(a)$ dans V telle que $\forall b \in A_v$, $cpt(a, b) = F$.

Si $var(a) = var(b)$ et pour toutes les variables $v \neq var(a)$, $\forall c \in A_v$, $cpt(a, c) = T \Rightarrow cpt(b, c) = T$, alors nous pouvons éliminer a de A par *substitution de voisinage*, car dans chaque solution dans laquelle a apparaît, nous pouvons remplacer a par b [12, 6]. Ni l'élimination d'une variable à une seule valeur, ni la cohérence d'arc, ni la substitution de voisinage ne peut introduire le motif interdit P quand l'opération est appliquée à une instance CSP($\overline{P}$). Pour simplifier nos preuves, nous supposons à partir de maintenant que nous avons appliqué ces trois opérations jusqu'à convergence à toutes les instances CSP.

Nous considérons maintenant deux nouvelles opérations de simplification. Ce sont des opérations de simplification qui peuvent s'appliquer à certaines instances CSP. Nous pouvons toujours effectuer la fusion de deux variables v_1, v_2 dans une instance CSP en une seule variable v dont l'ensemble des affectations est le produit cartésien des ensembles d'affectations à v_1 et à v_2. Sous certaines conditions, nous n'avons pas besoin de garder tous les éléments du produit cartésien et, en effet, le nombre total d'affectations en fait diminue.

Définition 4. *Considérons une instance CSP $\langle V, A, var, E, cpt\rangle$ avec $v_1, v_2 \in V$. Supposons qu'il y ait une* fonction de fusion $f : A_{v_1} \to A_{v_2}$, *telle que $\forall u \in A_{v_1}$, à chaque fois que u est dans une solution S, il y a une solution S' contenant à la fois u et $f(u)$. Alors nous pouvons effectuer la* fusion simple *de v_2 et v_1 pour créer une nouvelle variable* fusionnée *v dont le nombre d'affectations $|A_v| = |A_{v_1}|$. L'instance qui en résulte est $\langle V', A', var', E', cpt'\rangle$ définie par $V' = (V \setminus \{v_1, v_2\}) \cup \{v\}$, $A' = A \setminus A_{v_2}$, $var'(u) = var(u)$ pour tout $u \in A' \setminus A_{v_1}$ et $var'(u) = v$ pour tout $u \in A_{v_1}$, $E' = \{(p, q) \in \binom{A'}{2} \mid var'(p) \neq var'(q)\}$, $cpt'(p, q) = cpt(p, q)$ si $p, q \in A' \setminus A_{v_1}$, $cpt'(u, q) = cpt(u, q) \wedge cpt(f(u), q)$ pour tout $u \in A_{v_1}$ et tout $q \in A' \setminus A_{v_1}$.*

Définition 5. *Considérons une instance CSP $\langle V, A, var, E, cpt\rangle$ avec $v_1, v_2 \in V$ et une* valeur charnière *$a \in A_{v_1}$. Supposons qu'il y ait une* fonction de fusion $f : A_{v_1} \setminus \{a\} \to A_{v_2}$, *telle que $\forall u \in A_{v_1} \setminus \{a\}$, à chaque fois que u est dans une solution S, il y a une solution S' contenant à la fois u et $f(u)$. Alors nous pouvons effectuer la* fusion complexe *de v_2 et v_1 pour créer une nouvelle variable* fusionnée *v dont le nombre d'affectations $|A_v| = |A_{v_1}| + |A_{v_2}| - 1$. L'instance qui en résulte est $\langle V', A', var', E', cpt'\rangle$ définie par $V' = (V \setminus \{v_1, v_2\}) \cup \{v\}$, $A' = A \setminus \{a\}$, $var'(u) = var(u)$ pour tout $u \in A' \setminus (A_{v_1} \cup A_{v_2})$ et $var'(u) = v$ pour tout $u \in (A_{v_1} \setminus \{a\}) \cup A_{v_2}$, $E' = \{(p, q) \in \binom{A'}{2} \mid var'(p) \neq var'(q)\}$, $cpt'(p, q) = cpt(p, q)$ si $p, q \in A' \setminus (A_{v_1} \cup A_{v_2})$, $cpt'(u, q) = cpt(u, q) \wedge cpt(f(u), q)$ pour tout $u \in A_{v_1} \setminus \{a\}$ et tout $q \in A' \setminus (A_{v_1} \cup A_{v_2})$, $cpt'(p, q) = cpt(a, q) \wedge cpt(p, q)$ pour tout $p \in A_{v_2}$ et tout $q \in A' \setminus (A_{v_1} \cup A_{v_2})$.*

Lemme 1. *Si I est une instance CSP et I' le résultat d'une fusion (simple ou complexe) de deux variables de I, alors I' est soluble ssi I est soluble.*

Démonstration. Nous ne donnons la preuve que dans le cas d'une fusion complexe, car une fusion simple peut être vue comme un cas particulier. Parmi les affectations dans le produit cartésien de A_{v_1} et A_{v_2}, il est suffisant, afin de préserver la solubilité, de ne garder que ceux de la forme (a, q) avec $q \in A_{v_2}$ ou de la forme $(u, f(u))$ avec $u \in A_{v_1} \setminus \{a\}$. Pour compléter la preuve, il suffit de remarquer que dans A' nous utilisons $q \in A_{v_2}$ pour représenter la paire d'affectations (a, q) et $u \in A_{v_1} \setminus \{a\}$ pour représenter $(u, f(u))$. □

La fusion préserve la solubilité et le nombre total d'affectations diminue d'au moins 1 (en fait, par $|A_{v_2}|$ dans le cas d'une fusion simple). Cependant, lors de la résolution d'instances $I \in$ CSP($\overline{P}$), une opération de fusion ne sera utile que si elle n'introduit pas le motif interdit P.

3 Réduction et motifs intraitables

Dans un motif P, un point a qui n'est lié que par une seule arête de compatibilité au reste de P est un *point pendant*. Si une instance arc-cohérente I ne contient pas le motif P alors elle ne contient pas le motif P' qui est équivalent à P dans lequel le point pendant a et l'arête de compatibilité correspondante ont été supprimés. Ainsi, pour identifier les motifs traitables nous n'avons besoin d'étudier que les motifs sans points pendants.

Définition 6. *Nous disons qu'un motif P peut être* réduit *à un motif P', et que P' est une* réduction *de P, si P' est isomorphe à un motif Q dans*

la fermeture transitive des trois opérations extension, collage et élagage appliquées à P, où élagage est l'opération suivante :

élagage *éliminer un point pendant et son arête de compatibilité correspondante de P transforme P en Q.*

Le lemme suivant découle immédiatement des définitions.

Lemme 2. *Soient P et Q deux motifs, tels que P peut être réduit à Q. Soit I une instance CSP satisfaisant la cohérence d'arc. Si Q apparaît dans I, alors P apparaît aussi dans I. Si Q est traitable, alors P est traitable. Si P est intraitable, alors Q est intraitable.*

Il s'ensuit que nous n'avons besoin d'étudier que les motifs qui ne peuvent pas être réduit à un motif traitable connu et qui ne sont pas la réduction d'un motif intraitable connu. Dans le reste de cette section nous prouvons quelques résultats qui sont essentiels pour la preuve de la dichotomie pour motifs à 2 contraintes que nous donnons dans la section suivante.

Lemme 3. *Soit P un motif tel qu'une contrainte dans P contient deux arêtes d'incompatibilité distinctes qui ne peuvent pas être collées (par identification de points). Alors P est intraitable.*

Démonstration. Soit P un motif tel qu'une contrainte de P contient deux arêtes d'incompatibilité non collables. Soit SAT1 l'ensemble d'instances SAT avec au plus une apparition de chaque variable dans chaque clause. SAT1 est trivialement équivalente à SAT qui est NP-Complet [5]. Il suffit de donner une réduction polynomiale de SAT1 vers $\mathrm{CSP}(\overline{P})$. Nous supposons que nous avons une instance SAT1 $I = \{V, S\}$ avec V un ensemble de variables $\{v_1, v_2, \ldots, v_n\}$ et S un ensemble de clauses $\{C_1, C_2, \ldots, C_k\}$ tel que chaque clause C_i est une disjonction de c_i littéraux $l_i^1 \vee \cdots \vee l_i^{c_i}$. Nous créons l'instance CSP I' suivante :

- $n + k$ variables $v'_1, \ldots, v'_{n+k}$.
- $\forall v'_i$ avec $1 \leq i \leq n$, deux points "v_i" et "$\overline{v_i}$" dans $A_{v'_i}$.
- $\forall v'_i$ avec $n + 1 \leq i \leq n + k$, c_{i-n} points $l_{i-n}^1, \ldots, l_{i-n}^{c_{i-n}}$ dans $A_{v'_i}$.
- $\forall 1 \leq i \leq k$, $\forall 1 \leq j \leq c_i$, une arête d'incompatibilité entre le point $l_i^j \in A_{v'_{n+i}}$ et l'apparition dans $A_{v'_1}, \ldots, A_{v'_n}$ du littéral $\overline{l_i^j}$.

Par construction, I' a une solution si et seulement si I a une solution. De plus, chaque fois qu'une arête d'incompatibilité apparaît dans une contrainte C, cette contrainte C est entre une variable CSP v'_i représentant la variable SAT1 v_i et une variable CSP v'_{n+j} représentant la clause SAT1 C_j. Comme v_i apparaît au plus une fois dans C_j, alors il n'y a qu'une seule arête d'incompatibilité dans C. Donc I' ne contient pas le motif P. Donc nous avons réduit SAT1 à $\mathrm{CSP}(\overline{P})$. □

Définition 7. *Etant donné un motif $P = \langle V, A, var, E, cpt\rangle$, une variable $v \in V$, et un point $a \in A_v$, nous disons que a est* explicitement compatible *(respectivement* explicitement incompatible*) s'il y a un point $b \in A$ tel que a est compatible avec b (respectivement tel que a est incompatible avec b).*

Le lemme suivant s'ensuit de la définition de collage.

Lemme 4. *Soit P un motif dans lequel aucune paire de points n'est collable. Alors pour chaque variable v dans P, il n'y a au plus qu'un point dans A_v qui n'est pas explicitement incompatible.*

Lemme 5. *Soit Z le motif sur deux variables v et v' (donné en Figure 1), avec des points $a, b \in A_v$ et des points $c, d \in A_{v'}$ tels que a est compatible avec à la fois c et d, b est compatible avec c et incompatible avec d. Z est intraitable.*

Démonstration. Comme 3-COLOURING est NP-complet [13], il suffit de donner une réduction polynomiale depuis 3-COLOURING vers $\mathrm{CSP}(\overline{Z})$, l'ensemble d'instances CSP dans lesquelles le motif Z n'apparaît pas.

Définissons la contrainte $R_{s,t} \subseteq \{1, 2, 3\}^2$ par

$$R_{s,t} = \{\langle u, v\rangle | (u = s \wedge v = t) \vee (u \neq s \wedge v \neq t)\}$$

Il est facile de vérifier que $R_{s,t}$ ne contient pas le motif Z. Considérons le gadget à 5 variables avec les variables v_i, v_j, u_1, u_2, u_3, chacune avec pour domaine $\{1, 2, 3\}$, et avec les contraintes $R_{k,k}$ sur les variables (v_i, u_k) $(k = 1, 2, 3)$ et les contraintes $R_{1+(k \bmod 3),k}$ sur les variables (u_k, v_j) $(k = 1, 2, 3)$. L'effet combiné de ces six contraintes est simplement d'imposer la contrainte $v_i \neq v_j$. Toute instance $\langle V, E\rangle$ de 3-COLOURING, vers $V = \{1, \ldots, n\}$, peut être réduite à une instance de $\mathrm{CSP}(\overline{Z})$ avec pour variables $v_1, \ldots, v_n$ en plaçant une copie

de ce gadget entre chaque paire de variables (v_i, v_j) telles que $\{i, j\} \in E$. Cette réduction est clairement polynomiale. □

Lemme 6. *Tout motif P avec 2 contraintes sur 3 variables dans lequel les deux contraintes contiennent une arête d'incompatibilité et deux arêtes partageant un point mais non collables est intraitable.*

Démonstration. Nous allons réduire CSP à CSP($\overline{P}$). Soit I une instance CSP. Pour chaque (v, w) dans I tel qu'il y a une contrainte non triviale entre v et w, nous introduisons deux nouvelles variables v' et w' telles que le domaine de v' est le même que le domaine de v, le domaine de w' est le même que le domaine de w. Nous ajoutons des contraintes d'égalité entre v et v', et entre w et w', et nous ajoutons entre v' et w' la même contrainte qu'il y avait entre v et w. Toutes les autres contraintes impliquant v' ou w' sont triviales. Nous remplaçons aussi la contrainte entre v et w par une contrainte triviale. Soit I' l'instance obtenue après que toutes ces opérations ont été effectuées sur I. Par construction, I' a une solution si et seulement si I a une solution.

Nous supposons maintenant que nous avons trois variables v_0, v_1 et v_2 dans I' telles qu'il y a des contraintes non triviales entre v_0 et v_1 et entre v_0 et v_2. Par construction, au moins une de ces contraintes est une contrainte d'égalité. Par définition, un point dans une contrainte d'égalité est compatible avec un et un seul point. Ainsi, P ne peut pas apparaître sur v_0, v_1 et v_2. Cette réduction polynomiale de CSP vers CSP($\overline{P}$) montre que P est intraitable. □

4 Dichotomie pour les motifs à 2 contraintes

Soit $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5\}$ l'ensemble de motifs donnés en Figure 2.

Aucun motif de $\mathcal{T}$ ne peut être réduit à un autre motif de $\mathcal{T}$. Comme nous allons le montrer, chaque T_i définit une classe traitable d'instances CSP binaires. Par exemple, T_4 définit une classe d'instances qui inclut en tant que sous-ensemble propre toutes les instances avec contraintes zéro-une-toutes [7], une généralisation des clauses 2SAT aux logiques multi valuées. Comme les motifs traitables à

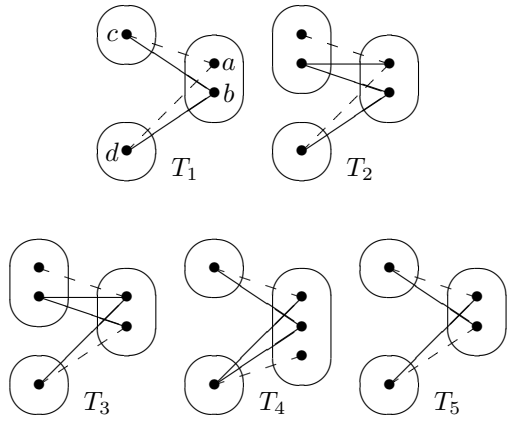


FIG. 2 – L'ensemble $\mathcal{T}$ des motifs traitables

2 contraintes sur 4 variables sont nécessairement composés de deux motifs triviaux à 1 contrainte, nous restreignons notre attention aux motifs à 2 contraintes sur 3 variables.

Théorème 1. *Soit P un motif à deux contraintes sur trois variables. Alors P est traitable si et seulement si P est réductible à un des motifs de $\mathcal{T}$.*

Démonstration. $\Rightarrow$: D'après le Lemme 3, nous savons que nous n'avons besoin d'étudier que les motifs avec au plus une arête d'incompatibilité dans chaque contrainte. Si l'une des contraintes ne contient aucune arête d'incompatibilité du tout, alors le motif est réductible par collage et/ou élagage à un motif avec une seule contrainte. Il n'est pas difficile de montrer que tous les motifs traitables à 1 contrainte sont réductibles à un des motifs de $\mathcal{T}$. Donc nous pouvons supposer à partir de maintenant qu'il y a exactement une arête d'incompatibilité $(a \in A_{v_0}, b \in A_{v_1})$ entre v_0 et v_1, et aussi exactement une arête d'incompatibilité $(c \in A_{v_0}, d \in A_{v_2})$ entre v_0 et v_2. Le "squelette" d'arêtes d'incompatibilité d'un motif traitable irréductible peut ainsi prendre deux formes selon si $a = c$ (squelette de type 1) ou $a \neq c$ (squelette de type 2).

D'après le Lemme 4 nous savons que $|A_v| \leq 2$ pour chaque variable v avec un seul point explicitement incompatible, et que $|A_v| \leq 3$ pour chaque variable v avec deux points explicitement incompatibles. Nous savons d'après les Lemmes 5 et 6 que tout motif à 2 contraintes sur 3 variables contenant Z ou dans lequel les deux contraintes contiennent chacune une arête d'incompatibilité et deux arêtes de compatibilité non collables est intraitable. Nous

savons que nous avons deux squelettes d'incompatibilité possibles à étudier, chacun impliquant un nombre maximum de points apparaissant dans le motif. Par une recherche exhaustive sur tous les motifs, nous pouvons déduire que tous les motifs traitables avec un squelette d'arêtes d'incompatibilité de type 1 sont réductibles par extension, collage et élagage à l'un de T_1 ou T_2, et que tous les motifs traitables avec un squelette d'arêtes d'incompatibilité de type 2 sont réductibles à l'un de T_3, T_4 ou T_5. Donc les seuls motifs traitables irréductibles possibles sont $T_1, \ldots, T_5$.

$\Leftarrow$: Nous donnons maintenant les preuves de polynomialité pour tous les motifs de $\mathcal{T}$.

Preuve de la polynomialité de T_1 : Considérons une instance de CSP($\overline{T_1}$).

Lemme 7. *Considérons une fusion (simple ou complexe) de deux variables v, v' dans une instance de* CSP($\overline{T_1}$)*. Supposons qu'à chaque fois que (a, a') et (b, b') sont des paires de points fusionnés durant cette fusion, telles que $a \neq b \in A_v$ et $a' \neq b' \in A_{v'}$, alors soit a et b' sont incompatibles ou b et a' sont incompatibles. Alors le motif T_1 ne peut pas être introduit par cette fusion.*

Démonstration. Par définition de la fusion (simple ou complexe), la seule façon d'introduire T_1 est quand les deux points dans la variable de droite de T_1 sont crées par la fusion de paires de points (a, a') et (b, b') telles que les compatibilités des points $a, b \in A_v$ et $a', b' \in A_{v'}$ avec les deux autres points c, d de T_1 sont données par : $cpt(c, a) = cpt(d, a') = F$, $cpt(c, b) = cpt(c, b') = cpt(a, a') = cpt(b, b') = cpt(d, b) = cpt(d, b') = T$. Maintenant, si a et b' étaient incompatibles, alors T_1 était déjà présent sur les points c, a, b, b' dans l'instance d'origine, et ne peuvent donc pas être introduits par la fusion. De même , si b et a' étaient incompatibles, alors T_1 était déjà présent sur les points b, a', b', d dans l'instance d'origine. □

Il est possible de montrer qu'après un prétraitement comprenant l'application d'opérations de fusion comme décrites dans le Lemme 7 jusqu'à convergence, nous avons une instance avec deux ensembles de variables E et $F = V \setminus E$ telle que :

- il n'y a pas de contrainte non triviale entre v et v' si $v, v' \in E$ ou $v, v' \in F$.
- $\forall v \in F, \forall f \in A_v$, f est incompatible avec des points dans $A_{v'}$ pour une et une seule variable $v' \in E$. De plus, f est incompatible avec tous les points de $A_{v'}$ sauf un.
- La seule contrainte non triviale possible entre une variable $v \in F$ et une variable $v' \in E$ est de la forme suivante : il y a un point $b \in A_v$ incompatible avec tous les points de $A_{v'}$ sauf un, et $\forall c \in A_v$ avec $c \neq b$, c est compatible avec tous les points de $A_{v'}$.

Le nombre total d'affectations diminue quand nous fusionnons des variables, donc le nombre total de fusions qui peuvent être effectuées est linéaire par rapport à la taille de l'instance d'origine.

Nous appelons SATNUF (pour SAT Non binaire Une seule Fois) le problème suivant : un ensemble de variables $V = \{v_1, v_2, \ldots, v_e\}$, un ensemble de valeurs $A = \{a_1, a_2, \ldots, a_n\}$, et un ensemble de clauses $C = \{C_1, C_2, \ldots, C_f\}$ tels que : chaque clause est une disjonction de littéraux, un littéral étant dans ce cas de la forme $v_i = a_j$, et $\forall i, j, p, q((v_i = a_j) \in C_p) \wedge ((v_i = a_j) \in C_q) \Rightarrow p = q$.

Lemme 8. *CSP($\overline{T_1}$) peut être réduit à SATNUF en temps polynomial.*

Démonstration. Nous supposons que nous avons une instance CSP binaire de CSP($\overline{T_1}$) et prétraitée comme décrit ci-dessus. Nous savons que les contraintes non triviales entre les variables $v \in F$ et $v' \in E$ sont toutes de la forme $v = b \Rightarrow v' = a$. De plus, chaque affectation $v = b$ d'une valeur à une variable apparaît dans exactement une de ces contraintes. Pour tout $v \in F$, nous pouvons remplacer l'ensemble de telles contraintes $v = b_i \Rightarrow v_i = a_i$, pour toutes les valeurs b_i dans le domaine de v, par la clause $(v_1 = a_1) \vee \ldots \vee (v_d = a_d)$. Il ne reste plus qu'a prouver qu'aucun littéral n'apparaît dans deux clauses distinctes. Supposons que nous avons un littéral $v_1 = a$ qui apparaît dans deux clauses distinctes. Alors il doit y avoir deux contraintes $v_2 = b \Rightarrow v_1 = a$ et $v_3 = c \Rightarrow v_1 = a$ et avec $v_1 \in E, v_2 \neq v_3 \in F$. Soit $a' \neq a$ un point de A_{v_1}. Alors b et c sont tous les deux incompatibles avec a' mais compatible avec a. Or c'est précisément le motif interdit. Cette contradiction montre que CSP($\overline{T_1}$) peut être réduit à SATNUF. □

Une instance CSP peut être vue comme une instance WCSP avec des fonctions de

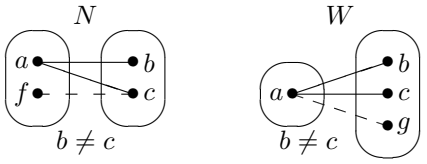


FIG. 3 – Deux gadgets

coût à valeurs dans $\{0, \infty\}$. En tant que fonctions de coût, les contraintes de SATNUF sont convexes, et les clauses sont non intersectantes. Donc, d'après [11], SATNUF est résoluble en temps polynomial, et par conséquent T_1 est traitable.

Preuve de la polynomialité de T_2 : Il est possible de montrer qu'après prétraitement, toute instance de $\mathrm{CSP}(\overline{T_2})$ se décompose en sous-instances indépendantes qui soit sont fonctionnelles [4] soit satisfont la propriété de gagnant ex-aequo ("Joint-winner property") [10], et par conséquent sont résolubles en temps polynomial.

Preuve de la polynomialité de T_3 : Considérons une instance de $\mathrm{CSP}(\overline{T_3})$.

Soit N le gadget donné en Figure 3 : deux variables v_0 et v_1 avec des points $a, f \in A_{v_0}$, $b, c \in A_{v_1}$, avec $b \neq c$, tels que a est compatible à la fois avec b et c, et f est incompatible avec c. Supposons que N apparaît dans l'instance et soit d un point dans A_{v_2}, avec $v_2 \neq v_0, v_1$. Si d est compatible avec c mais pas avec b, alors nous avons le motif interdit T_3. Donc si c est compatible avec un point hors de A_{v_0}, alors b est aussi compatible avec le même point.

Soit S une solution contenant c. Soit e le point de S dans A_{v_0}. Si e est compatible avec b, alors nous pouvons remplacer c par b dans S tout en maintenant la correction de la solution, car tous les points de l'instance hors de A_{v_0} qui sont compatibles avec c sont aussi compatibles avec b.

Si e n'est pas compatible avec b, alors les arêtes $\{b, e\}$, $\{e, c\}$ et $\{c, a\}$ forment le gadget N. Donc, par notre argument précédent, si e est compatible avec un point hors de A_{v_1}, alors a est aussi compatible avec le même point. Nous pouvons donc remplacer c par b et e par a dans S tout en maintenant la correction de la solution, car tous les points de l'instance hors de A_{v_0} qui sont compatibles avec c sont aussi compatibles avec b et tous les points de l'instance hors de A_{v_1} qui sont compatibles avec e sont aussi compatibles avec a. Donc si une solution contient c, alors il y a une autre solution contenant b. Ainsi nous pouvons supprimer c tout en préservant la solubilité.

Donc à chaque fois que le gadget N est présent, nous pouvons supprimer l'un de ses points et ainsi éliminer N. Le gadget N est un motif traitable connu car interdire N revient à dire que toutes les contraintes sont soit triviales soit des bijections. Donc s'il n'est pas présent, alors l'instance est traitable. Il s'ensuit que le motif T_3 est traitable.

Preuve de la polynomialité de T_4 : Considérons une instance de $\mathrm{CSP}(\overline{T_4})$.

Soit W le gadget donné en Figure 3 : deux variables v_0 et v_1 telles que $a \in A_{v_0}$, $b, c, g \in A_{v_1}$, avec $b \neq c$, a compatible avec à la fois b et c, et a incompatible avec g. Supposons que nous avons W dans l'instance.

Soit f un point dans A_{v_2}, avec $v_2 \neq v_0, v_1$. Si f est compatible avec b et incompatible avec c (ou compatible avec c et incompatible avec b), alors nous avons le motif interdit T_4. Donc, tous les points de l'instance hors de $A_{v_0} \cup A_{v_1}$ ont la même compatibilité avec b et c.

Si tous les points de A_{v_0} qui sont compatibles avec b sont aussi compatibles avec c, alors tous les points de l'instance compatibles avec b sont aussi compatible avec c et on peut supprimer b par la substitution de voisinage. Donc, nous pouvons supposer qu'il existe $d \in A_{v_0}$ tel que d est compatible avec b et incompatible avec c.

Soit S une solution qui contient c. Soit e le point de S dans v_0. Si e est compatible avec b, alors on peut remplacer c par b dans S tout en conservant la correction de la solution, car b et c ont la même compatibilité avec tous les points de l'instance hors de $A_{v_0} \cup A_{v_1}$. Si e n'est pas compatible avec b, alors les arêtes $\{b, e\}$, $\{b, a\}$ et $\{b, d\}$ constituent le gadget W. Donc, par l'argument ci-dessus, a et d ont la même compatibilité avec tous les points de l'instance hors de $A_{v_0} \cup A_{v_1}$. De même, les arêtes $\{c, d\}$, $\{c, a\}$ et $\{c, e\}$ constituent le gadget W. Donc a et e ont la même compatibilité avec tous les points de l'instance hors de $A_{v_0} \cup A_{v_1}$. Ainsi d et e ont la même compatibilité avec tous les points de l'instance hors de $A_{v_0} \cup A_{v_1}$. Donc on peut remplacer c par b et e par d dans S tout en conservant la correction de la solution, car b,c ont la même compatibilité avec tous les

points de l'instance hors de $A_{v_0} \cup A_{v_1}$ et e,d ont la même compatibilité avec tous les points de l'instance hors de $A_{v_0} \cup A_{v_1}$.

Comme dans la preuve de polynomialité de T_3, nous avons montré que si une solution contient c, alors il y a une autre solution contenant b. Ainsi nous pouvons supprimer c. Par conséquent, à chaque fois que le gadget W est présent, nous pouvons supprimer l'un de ses points. Le gadget W est un motif traitable connu car interdire W revient à dire que toutes les contraintes sont zéro-une-toutes [7]. Donc s'il n'est pas présent, l'instance est traitable. Par conséquent le motif T_4 est traitable.

Preuve de la polynomialité de T_5 : Le motif T_5 est un sous-motif du motif du triangle-cassé BTP, un motif traitable connu [9] sur trois contraintes. Donc le motif T_5 est traitable. □

5 Conclusion

Nous avons démontré une dichotomie pour des classes d'instance CSP binaires définies par l'interdiction de motifs à 2 contraintes. Cela nous a permis d'identifier de nouvelles classes traitables, comprenant, par exemple, une nouvelle généralisation des contraintes zéro-une-toutes. Une voie de recherche future est d'examiner les généralisations possibles des cinq classes traitables définies par les motifs interdits $T_1, \ldots, T_5$, en remplaçant les contraintes binaires par des contraintes k-aires ($k > 2$) ou en ajoutant d'autres contraintes aux motifs.

Nous avons aussi identifié des opérations de prétraitement, basées sur la fusion de deux variables, qui pourraient s'appliquer à des instances CSP quelconques.

Références

[1] Bessière, C., Régin, J.-C., Yap, R. H. C. and Zhang, Y., An optimal coarse-grained arc consistency algorithm, *Artificial Intelligence*, 165 (2) (2005) 165–185.

[2] Bulatov, A., Jeavons, P. and Krokhin, A., Classifying the complexity of constraints using finite algebras, *SIAM Journal on Computing*, 34 (3) (2005) 720–742.

[3] Bulatov, A. A., A dichotomy theorem for constraint satisfaction problems on a 3-element set, *J. of the ACM*, 53(1) (2006) 66–120.

[4] Cohen, D. A., Cooper, M .C., Green, M. J. and Marx, D., On guaranteeing polynomially bounded search tree size, *CP'11*, LNCS 6876 (2011) 160–171.

[5] Cook, S. A., The Complexity of Theorem-Proving Procedures, *STOC : Proceedings of the ACM symposium on Theory of computing* (1971) 151–158.

[6] Cooper, M. C. Fundamental properties of neighbourhood substitution in constraint satisfaction problems, *Artificial Intelligence* 90 (1997) l–24.

[7] Cooper, M. C., Cohen, D. A. and Jeavons, P. G. Characterising tractable constraints, *Artificial Intelligence* 65 (2) (1994) 347–361.

[8] Cooper, M. C. and Escamocher, G., A dichotomy for 2-constraint forbidden CSP patterns, *CoRR* abs/1201.3868 (2012).

[9] Cooper, M. C., Jeavons, P. G., and Salamon, A. Z., Generalizing constraint satisfaction on trees : Hybrid tractability and variable elimination, *Artificial Intelligence*, 174 (9–10) (2010) 570–584.

[10] Cooper, M. C. and Živný, S., Hybrid tractability of valued constraint problems, *Artificial Intelligence*, 175 (9-10) (2011) 1555–1569.

[11] Cooper, M. C. and Živný, S., Hierarchically nested convex VCSP, *CP 2011*, LNCS 6876 (2011) 187–194.

[12] Freuder, E. C., Eliminating interchangeable values in constraint satisfaction problems, *AAAI-91*, Anaheim, CA (1991) 227–233.

[13] Garey, M. R. and Johnson, D. S., *Computers and Intractability : A Guide to the Theory of NP-Completeness*, W.H. Freeman (1979).

[14] Grohe, M., The complexity of homomorphism and constraint satisfaction problems seen from the other side, *J. of the ACM*, 54 (1) (2007) 1–24.

[15] Marx, D., Can You Beat Treewidth ?, *Theory of Computing*, 6 (1) (2010) 85–112.

[16] Marx, D., Tractable hypergraph properties for constraint satisfaction and conjunctive queries, *STOC '10* (2010) 735–744.

Une classe traitable de problèmes de Planification temporelle

Martin C. Cooper* **Frédéric Maris*** **Pierre Régnier*** **Florian Franc***

IRIT, Université de Toulouse
Toulouse, France
{cooper, maris, regnier}@irit.fr

Résumé

Cet article présente une classe de problèmes de planification temporelle solubles en temps polynomial. Ce résultat découle de deux hypothèses. Nous supposons d'abord que les sous-buts ne peuvent être établis que par une action unique, ce qui nous permet de déterminer rapidement les actions qui sont nécessaires dans tous les plans. Nous supposons également que les sous-buts sont monotones, ce qui nous permet d'exprimer la planification comme une instance de STP$^{\neq}$ (Simple Temporal Problem, difference constraints). Notre classe contient des problèmes temporellement expressifs, ce que nous illustrons avec un exemple de planification de processus chimique.

1 Introduction

La planification est un domaine de l'IA qui, dans le cas général, n'est pas soluble en temps polynomial (on dit encore non traitable) [11]. [5] montre en particulier que la planification propositionnelle est PSPACE-complète.

Malgré ces résultats généraux peu encourageants, de nombreux travaux ont été réalisés pour étudier la complexité de la planification non-optimale et optimale des benchmarks classiques. [18], [19], [30] ont ainsi montré qu'un bon nombre de ces problèmes pouvaient être résolus par des procédures simples s'exécutant en temps polynomial. Les planificateurs FF [20] et eCPT [31] ont aussi prouvé (bien que de manière empirique) que le nombre de benchmarks qui pouvaient être résolus sans recherche devait être bien plus important.

A partir des travaux de [1] sur le formalisme de planification SAS (Simplified Action Structure), un bon nombre d'études ont également été réalisées afin de mettre en évidence des classes de problèmes de planification traitables. Une bonne partie des résultats obtenus est basée sur des restrictions syntaxiques qui portent sur l'ensemble des opérateurs [5], [2], [11], [25]. Un autre ensemble de travaux utilise la structure sous-jacente des problèmes de planification qui peut être mise en évidence grâce au graphe de causalité [27]. En imposant des restrictions sur la structure de ce graphe, on peut exhiber des classes traitables [23], [24], [25], [32], [10], [3], [4], [18], [19], [22], [16], [17], [15], [26]. Un cadre unificateur permettant de classifier la complexité de la planification sous restrictions sur le graphe de causalité est donné dans [6].

Malgré ces nombreux résultats, les hypothèses de la planification classique (monde fini, statique, actions instantanées, déterministes…), sont bien trop restrictives pour permettre de modéliser des applications réelles. Pour cela, il est nécessaire d'utiliser un cadre de planification temporelle afin de pouvoir formaliser les relations temporelles entre actions comme des contraintes temporelles. Cependant, dans le cadre temporel de PDDL 2.1 [9], [12], tester l'existence d'un plan valide devient un problème EXPSPACE-complet [29]. La complexité PSPACE-complète de la planification classique ne peut être préservée que lorsque différentes instances d'une même action ne peuvent se chevaucher.

Dans cet article, nous présentons un sous-problème de la planification temporelle soluble en temps polynomial. A notre connaissance, aucun travail antérieur n'a spécifiquement abordé cette question.

L'article est organisé ainsi : la Section 2 présente notre cadre temporel. La section suivante introduit ensuite la notion de monotonie des fluents (propositions atomiques). La Section 4 montre comment déterminer certains fluents monotones. La Section 5 donne un exemple d'un problème de planification temporelle (Temporal Chemical Process Planning) soluble en temps polynomial. Toutes les solutions de cet exemple nécessitent la concurrence des actions. Les Sections 6 et 7 concluent et présentent nos perspectives de recherche.

* Les auteurs sont soutenus par le Projet ANR-10-BLAN-0210.

2 Planification Temporelle

Nous étudions la planification temporelle propositionnelle dans un langage basé sur les aspects temporels de PDDL 2.1. Un *fluent* est une proposition atomique positive ou négative. Comme dans le langage PDDL 2.1, nous considérons que les changements de valeur des fluents sont instantanés mais que les conditions sur leurs valeurs peuvent être imposées sur un intervalle. Une *action a* est un quadruplet <Cond(*a*), Add(*a*), Del(*a*), Constr(*a*)>, où l'ensemble des conditions Cond(*a*) est l'ensemble des fluents qui doivent nécessairement être vrais pour que *a* soit exécutée, l'ensemble des ajouts Add(*a*) est l'ensemble des fluents qui sont établis par *a*, l'ensemble des retraits Del(*a*) est l'ensemble des fluents qui sont détruits par *a*, et l'ensemble des contraintes Constr(*a*) est l'ensemble des contraintes entre les instants relatifs aux événements qui interviennent durant l'exécution de *a*. Un événement correspond à l'une des quatre possibilités suivantes : l'établissement ou la destruction d'un fluent par une action *a*, ou le début ou la fin d'un intervalle sur lequel un fluent est requis par *a*. En PDDL 2.1, les événements ne peuvent intervenir qu'en début ou fin des actions, mais nous relaxons cette hypothèse pour qu'ils puissent survenir à tout instant à condition que les contraintes Constr(*a*) soient satisfaites.

Nous utilisons la notation $a \rightarrow f$ pour dénoter l'événement qui correspond à l'établissement du fluent *f* par l'action *a*, la notation $a \rightarrow \neg f$ pour dénoter l'événement qui correspond à la destruction du fluent *f* par l'action *a*, et les notations $f \mid\!\rightarrow a$ et $f \rightarrow\!\mid a$, respectivement, pour dénoter le début et la fin de l'intervalle sur lequel *a* nécessite la condition *f*. Si *f* est déjà vrai (respectivement faux) quand l'événement $a \rightarrow f$ ($a \rightarrow \neg f$) intervient, nous considérons toujours que *a* établit (détruit) *f*. Un plan temporel peut contenir plusieurs instances de la même action, mais puisque la plupart des plans temporels étudiés dans cet article contiennent au plus une instance de chaque action, pour simplifier les notations, nous ne ferons la distinction entre actions et instances d'actions que lorsque cela sera absolument nécessaire. Nous utilisons la notation $\tau(E)$ pour représenter l'instant auquel un événement E intervient dans un plan.

Pour une action *a* donnée, Events(*a*) représente les différents événements qui le définissent, à savoir ($a \rightarrow f$) pour tout *f* dans Add(*a*), ($a \rightarrow \neg f$) pour tout *f* dans Del(*a*), ($f \mid\!\rightarrow a$) et ($f \rightarrow\!\mid a$) pour tout *f* dans Cond(*a*). La définition d'une action *a* contient des contraintes Constr(*a*) sur les instants relatifs aux événements de Events(*a*). Comme en PDDL 2.1, nous considèrerons que le délai entre événements dans Events(*a*) n'est pas nécessairement fixé et que Constr(*a*) correspond à des contraintes d'intervalles sur des paires d'événements, telles que $\tau(f \rightarrow\!\mid a) - \tau(f \mid\!\rightarrow a) \in [\alpha, \beta]$ pour des constantes α,β. Nous utilisons $[\alpha_a(E_1,E_2), \beta_a(E_1,E_2)]$ pour dénoter l'intervalle des valeurs possibles pour la distance relative entre les événements E_1,E_2 dans l'action *a*. Un délai fixe entre les événements $E_1,E_2 \in$ Events(*a*) peut aussi être modélisé en choisissant $\alpha_a(E_1,E_2) = \beta_a(E_1,E_2)$. Nous introduisons maintenant deux contraintes de base que tous les plans temporels doivent vérifier.

Contraintes inhérentes sur l'ensemble des actions A : pour toute $a \in$ A, *a* satisfait Constr(*a*), i.e. pour toute paire d'événements $E_1,E_2 \in$ Events(a), $\tau(E_1) - \tau(E_2) \in [\alpha_a(E_1,E_2), \beta_a(E_1,E_2)]$.

Contraintes d'effets contradictoires sur l'ensemble des actions A : pour toutes les actions $a_i,a_j \in$ A, pour tout fluent positif $f \in$ Del(a_i) ∩ Add(a_j), $\tau(a_i \rightarrow \neg f) \neq \tau(a_j \rightarrow f)$.

Définition 1. Un *problème de planification temporelle* <I,A,G> est défini par un ensemble d'actions A, un état initial I et un but G, où I et G sont des ensembles de fluents.

Notation: Si A est un ensemble d'instances d'actions, alors Events(A) est l'union des ensembles Events(*a*) (pour toutes les instances d'actions $a \in$ A).

Définition 2. P = <A,τ>, avec A un ensemble d'instances d'actions $\{a_1,...,a_n\}$ et τ une fonction réelle sur Events(A), est un *plan temporel* pour le problème <I,A′,G> si
(1) A ⊆ A′, et
(2) P satisfait les contraintes inhérentes et d'effets contradictoires sur A;

et si, lorsque P est executé (i.e. les fluents sont établis ou détruits aux instants donnés par τ), en démarrant de l'état initial I :
(3) pour toute $a_i \in$ A, chaque $f \in$ Cond(a_i) est vrai lorsqu'il est requis, et
(4) tous les fluents du but $g \in$ G sont vrais à la fin de l'exécution de P.

Etudions maintenant plus en détail le type de contraintes que nous imposons sur les instants relatifs aux événements à l'intérieur d'une action.

Définition 3. Une *contrainte d'intervalle* C(x,y) sur des variables réelles x,y est une contrainte de la forme $x\text{-}y \in [a,b]$ où a,b sont des constantes réelles.

Définition 4. [21] Une contrainte binaire C(x,y) est *min-closed* si pour toute paire de valeurs (x_1,y_1), (x_2,y_2) qui satisfait C, $(\min(x_1,x_2), \min(y_1,y_2))$ satisfait également C.

Lemme 1. Soit A=$\{a_1,...,a_n\}$ un ensemble d'actions et A′ un ensemble d'instances d'actions dans lequel chaque action a_i (i=1,...,n) intervient $t_i \geq 1$ fois. Soit τ une fonction réelle sur l'ensemble des événements dans A′. Pour chaque E ∈ Events(a_i), soit E[j] (j=1,...,t_i) l'occurrence de l'événement E dans la $j$ème instance de a_i. Pour $i \in \{1,...,n\}$, on définit la fonction réelle τ_{min} sur l'ensemble des événements dans l'ensemble des actions A par : $\tau_{min}(E) = \min\{\tau(E[j]) \mid j=1,...,t_i\}$. Si τ satisfait les contraintes inhérentes sur A′, alors τ_{min} satisfait les contraintes inhérentes sur A.

Preuve : Toutes les contraintes d'intervalle sont min-closed [21]. En appliquant la définition de min-closed t_i–1 fois, pour chaque action a_i, nous pouvons déduire que si τ satisfait une contrainte d'intervalle sur chacune des t_i instances de a_i, alors τ_{min} satisfait cette contrainte sur l'action a_i. En d'autres termes, pour toutes les paires d'événements E_1,E_2 dans Events(a_i), si $\tau(E_1[j]) - \tau(E_2[j]) \in [\alpha_a(E_1,E_2), \beta_a(E_1,E_2)]$ pour j=1,...,t_i, alors $\tau_{min}(E_1) - \tau_{min}(E_2) \in [\alpha_a(E_1,E_2), \beta_a(E_1,E_2)]$. Donc si τ satisfait les contraintes inhérentes sur A′, alors τ_{min} satisfait les contraintes inhérentes sur A.

Définition 5. Un problème de planification temporelle <I,A,G> est *positif* s'il n'y a aucun fluent négatif ni dans les conditions des actions, ni dans le but G.

Dans cet article, nous considèrerons uniquement des problèmes de planification temporelle positifs <I,A,G>. Il est bien connu que tout problème de planification peut être transformé en un problème positif équivalent en temps linéaire par introduction d'un nouveau fluent *notf* pour chaque fluent négatif *f* [14]. Sous cette hypothèse, G et Cond(*a*) (pour toute action *a*) sont composés de fluents positifs. Par convention, Add(*a*) et Del(*a*) sont aussi composés exclusivement de fluents positifs. L'état initial I peut contenir des fluents négatifs.

Pour définir notre classe traitable de problèmes de planification, nous aurons besoin de définir la notion d'ensemble d'actions *establisher-unique*. Cette notion est équivalente à celle de *post-unique* de la planification SAS^+ [25] restreinte aux variables booléennes mais généralisée pour qu'elle s'applique à un sous-ensemble spécifique des fluents positifs. Dans la section suivante, nous l'appliquons à l'ensemble des sous-buts S, qui sont les fluents essentiels à la réalisation du but.

Définition 6. Un ensemble d'actions A={a_1,...,a_n} est *establisher-unique* relativement à un ensemble de fluents positifs S si pour tout $i \neq j$, Add(a_i) $\cap$ Add(a_j) $\cap$ S = $\varnothing$, i.e. aucun fluent de S ne peut être établi par deux actions distinctes de A.

Si un ensemble d'actions est establisher-unique relativement à l'ensemble des sous-buts d'un problème, alors nous pouvons déterminer en temps polynomial l'ensemble des actions qui sont nécessairement présentes dans un plan temporel. Pour produire un plan temporel valide il reste encore à déterminer combien de fois chaque action doit intervenir et à ordonner ces instances d'actions.

3 Planification Monotone

Dans cette section, nous introduisons la propriété de monotonie des fluents. Avec la notion d'ensemble d'actions establisher-unique, la monotonie des fluents est une condition suffisante pour montrer qu'il existe un algorithme polynomial pour la planification temporelle.

Définition 7. Un fluent *f* est *–monotone* (relativement à un probleme de planification temporelle positif <I,A,G>) si, après avoir été détruit, *f* n'est jamais ré-établi dans aucun plan temporel pour <I,A,G>. Ce fluent est *+monotone* si, après avoir été établi, il n'est jamais détruit dans aucun plan temporel pour <I,A,G>. Un fluent est *monotone* (relativement à <I,A,G>) s'il est *+monotone* ou *–monotone* (relativement à <I,A,G>).

Exemples : Dans des contextes évidents, les fluents suivants sont –monotones : vivante (une personne), jamais-utilisée, allumée (une allumette), prêt-à-manger (un plat). De même, les fluents suivants sont tous +monotones : non-vivante, brulée, dissous, explosé (un ballon), mangé, cuisiné, diplômé, né et éteint. La manière de détecter la monotonie des fluents est discutée dans la Section 4.

Notation : Si A est un ensemble d'actions, nous utilisons la notation Del(A) pour représenter l'union des ensembles Del(*a*) (pour toutes les actions $a \in$ A). Add(A), Cond(A), Constr(A) sont définies de façon similaire.

Le lemme suivant résulte trivialement de la Définition 7.

Lemme 2. Si $f \notin$ Add(A) $\cap$ Del(A), alors *f* est à la fois –monotone et +monotone relativement au problème de planification temporelle positif <I,A,G>.

Nous introduisons maintenant trois autres ensembles de contraintes qui ne s'appliqueront qu'aux fluents monotones :

Contraintes de –autorisation sur les fluents positifs *f* et l'ensemble d'actions A : pour toutes les actions $a_i,a_j \in$ A, si $f \in$ Del(a_j) $\cap$ Cond(a_i), alors $\tau(f \rightarrow\mid a_i) < \tau(a_j \rightarrow \neg f)$.

Contraintes de +autorisation sur les fluents positifs *f* et l'ensemble d'actions A : pour toutes les actions $a_i,a_j \in$ A, si $f \in$ Del(a_j) $\cap$ Add(a_i) $\cap$ (Cond(A) $\cup$ G), alors $\tau(a_j \rightarrow \neg f) < \tau(a_i \rightarrow f)$.

Contraintes de causalité sur les fluents positifs *f* et l'ensemble d'actions A : pour toutes les actions $a_i,a_j \in$ A, si $f \in$ (Cond(a_j) $\cap$ Add(a_i))\I alors $\tau(a_i \rightarrow f) < \tau(f \mid\rightarrow a_j)$.

Définition 8. Un plan temporel pour un problème de planification temporelle positif <I,A,G> est *monotone* si chaque paire d'actions (dans A) satisfait les contraintes de +autorisation pour tous les fluents +monotones et les contraintes de –autorisation pour tous les fluents –monotones.

Le lemme suivant résulte directement de la définition de la monotonie et du fait qu'un fluent ne peut être simultanément établi et détruit dans un plan temporel.

Lemme 3. Supposons qu'un fluent positif *f* est monotone relativement à un problème de planification temporelle positif <I,A,G> et que les actions $a_i,a_j \in$ A sont telles que $f \in$ Add(a_i) $\cap$ Del(a_j). Si *f* est +monotone relativement à

ce problème, alors dans tous les plans temporels contenant à la fois a_i et a_j, $\tau(a_j \rightarrow \neg f) < \tau(a_i \rightarrow f)$. Si f est –monotone relativement à ce problème, alors dans tous les plans temporels contenant à la fois a_i et a_j, $\tau(a_i \rightarrow f) < \tau(a_j \rightarrow \neg f)$.

Proposition 1. Si chaque fluent dans Cond(A) est monotone relativement à un problème de planification temporelle positif <I,A,G>, alors tous les plans temporels pour <I,A,G> sont monotones.

Preuve : Soit P un plan temporel. Considérons d'abord un fluent positif –monotone f. Nous devons montrer que les contraintes de –autorisation sont satisfaites pour f dans P, i.e. que f n'est pas détruit avant qu'il soit requis dans P (ou au même instant). Mais ceci doit être le cas puisque, une fois détruit, f ne peut être ré-établi. Considérons ensuite un fluent positif +monotone f. Par le Lemme 3, les contraintes de +autorisation sont satisfaites pour f dans P.

Définition 9. Etant donné un problème de planification temporelle <I,A,G>, l'*ensemble des sous-buts* est le sous-ensemble minimal SG de Cond(A) ∪ G qui satisfait :
(1) G ⊆ SG et
(2) pour toute action a ∈ A, si Add(a) ∩ (SG\I) ≠ ∅, alors Cond(a) ⊆ SG.

L'*ensemble d'actions réduit* est $A^r = \{a \in A \mid Add(a) \cap (SG\backslash I) \neq \emptyset\}$.

Il est évident que nous pouvons déterminer SG et A^r en temps polynomial et le résultat est unique. Si chaque fluent dans Cond(A^r) ∪ G est monotone, nous pouvons dire qu'un plan P pour le problème de planification temporelle <I,A,G> *satisfait les contraintes d'autorisation* si chaque fluent –monotone satisfait les contraintes de –autorisation et chaque fluent +monotone satisfait les contraintes de +autorisation (en partant de l'hypothèse que nous savons, pour chaque fluent $f \in$ Cond(A^r) ∪ G, si f est + ou –monotone).

Théoreme 1. Etant donné un problème de planification temporelle positif <I,A,G>, où A est un ensemble d'actions tel que Constr(A) sont des contraintes d'intervalle, soit SG et A^r, respectivement, l'ensemble des sous-buts et l'ensemble d'actions réduit. Si l'ensemble d'actions A^r est establisher-unique relativement à SG, chaque fluent dans Cond(A^r) ∪ G est monotone relativement à <I,A^r,G> et chaque fluent dans I ∩ (Cond(A^r) ∪ G) est –monotone relativement à <I,A^r,G>, alors <I,A,G> a un plan temporel P si est seulement si :
(1) G ⊆ (I\Del(A^r)) ∪ Add(A^r)
(2) Cond(A^r) ⊆ I ∪ Add(A^r)
(3) tous les fluents $g \in$ G ∩ Del(A^r) ∩ Add(A^r) sont +monotones relativement à <I,A^r,G>
(4) l'ensemble des contraintes d'autorisation, inhérentes, d'effets contradictoires et de causalité peut être satisfait sur l'ensemble d'actions A^r.

Preuve : (⇒) A^r est l'ensemble des actions qui établissent les sous-buts f dans SG\I. SG = Cond(A^r) ∪ G. Puisque A^r est establisher-unique relativement à SG, chaque sous-but $f \in$ SG\I est établi par une action unique. Chaque action dans A^r doit donc intervenir dans le plan P. Par ailleurs, (Add(A)\Add(A^r)) ∩ (Cond(A^r)\I) = ∅ par la Définition 9. Il résulte que (2) est une condition nécessaire pour qu'un plan temporel P existe.

Soit P′ une version de P dans laquelle nous ne conservons que les actions de A^r. P′ est également un plan temporel valide puisqu'aucune condition des actions dans P′ et aucun but dans G ne sont établis par une des actions éliminées de P, sauf peut-être s'il sont également dans I. Ces fluents $f \in$ I ∩ (Cond(A^r) ∪ G) sont –monotones, par hypothèse, et ne peuvent donc être établis dans P après avoir été détruits. Il en résulte que tous ces f sont déjà vrais lorsqu'ils sont établis dans P par toute action dans A\ A^r.

(1) est clairement une condition nécessaire pour que P′ soit un plan valide. Considérons $g \in$ G ∩ Del(A^r) ∩ Add(A^r). Par le Lemme 3, nous pouvons déduire que g ne peut être –monotone, puisque g est vrai à la fin de l'exécution de P′. (3) est ainsi vérifiée. Soit P_{min}=<A^r,τ_{min}> la version du plan temporel P′=<A^r,τ> dans lequel nous ne conservons qu'une seule instance de chaque action $a_i \in A^r$ (et pas d'instances des actions dans A\A^r) et τ_{min} est définie à partir de τ en prenant la première instance de chaque événement dans Events(a_i), pour chaque action $a_i \in A^r$, comme décrit dans l'énoncé du Lemme 1. Nous allons montrer que P_{min} satisfait les contraintes d'autorisation, inhérentes, d'effets contradictoires et de causalité.

D'après la Proposition 1, le plan temporel P′ doit être monotone. Puisque P′ est monotone et d'après la définition d'un plan temporel, les contraintes d'autorisation sont toutes satisfaites. P′ doit également, d'après la définition d'un plan temporel, satisfaire les contraintes inhérentes et d'effets contradictoires. Il résulte du Lemme 1 que P_{min} satisfait également les contraintes inhérentes. Puisque les événements dans P_{min} sont simplement un sous-ensemble des événements dans P′, P_{min} satisfait nécessairement les contraintes d'autorisation et les contraintes d'effets contradictoires.

Considérons un fluent positif $f \in$ (Cond(a_j) ∩ Add(a_i))\I, où a_i, $a_j \in A^r$. Puisque $a_j \in A^r$, nous savons que Add(a_j) ∩ (SG\I) ≠ ∅ et donc que Cond(a_j) ⊆ SG, d'après la définition de l'ensemble des sous-buts SG. Puisque $f \in$ Cond(a_j) nous pouvons déduire que $f \in$ SG. En fait, $f \in$ SG\I puisque nous supposons que $f \notin$ I. Il en résulte que si $f \in$ Add(a) pour une action $a \in$ A, alors $a \in A^r$. Mais nous savons que A^r est establisher-unique (relativement à SG). Donc, puisque $f \in$ Cond(a_j) ⊆ Cond(A^r) et $f \in$ Add(a_i), f peut être établi par la seule action a=a_i dans A. Puisque $f \notin$ I, le premier établissement de f par une instance de a_i doit intervenir dans P′ avant que f soit requis par n'importe quelle instance de a_j. Il en

résulte que les contraintes de causalité sont satisfaites par f dans P_{min}.

($\Leftarrow$) Supposons que (1) et (2) sont satisfaites par A^r. Soit P une solution à l'ensemble des contraintes d'autorisation, inhérentes, d'effets contradictoires et de causalité sur A^r. Une solution à ces contraintes utilise chaque action dans A^r (en fait, elle utilise chaque action exactement une fois puisque elle assigne un instant de démarrage à chaque action dans A^r). Considérons un fluent quelconque $g \in G$. Par (1), $g \in (I \backslash Del(A^r)) \cup Add(A^r)$. Si $g \notin Del(A^r)$, alors g est nécessairement vrai à la fin de l'exécution de P. D'un autre côté, si $g \in Del(a_j)$ pour certaines actions $a_j \in A^r$, alors par (1) il y a nécessairement une action $a_i \in A^r$ qui établit g. Alors, par (3) g est +monotone. Puisque P satisfait les contraintes de +autorisation pour g, a_i établit g après toutes les destructions de g. Il en résulte que g est vrai à la fin de l'exécution de P.

Considérons des fluents –monotone $f \in Cond(a_j)$ où $a_j \in A^r$. Puisque les contraintes de –autorisation sont satisfaites pour f dans P, f peut seulement être détruit dans P après qu'il soit requis par a_j. Il ne reste donc plus à montrer que f était soit vrai dans l'état initial I ou qu'il a été établi à un instant qui précède celui auquel il est requis par a_j. Par (2), $f \in I \cup Add(A^r)$, et nous n'avons besoin de considérer que le cas dans lequel $f \notin I$ mais $f \in Add(a_i)$ pour une action $a_i \in A^r$. Puisque P satisfait les contraintes de causalité, $\tau(a_i \rightarrow f) < \tau(f \mid\rightarrow a_j)$ et, durant l'exécution de P, f est donc vrai quand il est requis par l'action a_j.

Considérons un fluent $f \in Cond(a_j)$, où $a_j \in A^r$, qui n'est pas –monotone. Par les hypothèses du théorème, f est nécessairement +monotone et $f \notin I$. Considérons d'abord le cas $f \notin Del(A^r) \cap Add(A^r)$. Par le Lemme 2, f est –monotone ce qui contredit notre hypothèse. Donc $f \in Del(a_k) \cap Add(a_i)$, pour des actions a_i, $a_k \in A^r$, et rappelons que $f \notin I$. Puisque les contraintes de +autorisation sont satisfaites pour f dans P, toute destruction de f intervient avant que f soit établi par a_i. Il résulte alors des contraintes de causalité que la condition f sera vraie quand elle sera requise par a_j pendant l'exécution de P.

Théorème 2. Soit Π^{U+M} la classe des problèmes de planification temporelle positifs <I,A,G> dans lesquels A est establisher-unique relativement à $Cond(A) \cup G$, tous les fluents dans $Cond(A) \cup G$ sont monotones relativement à <I,A,G> et tous les fluents dans $I \cap (Cond(A) \cup G)$ sont –monotones relativement à <I,A,G>. Alors Π^{U+M} peut être résolue en temps $O(n^3)$ et en espace $O(n^2)$, où n est le nombre total d'événements dans les actions de A. En effet, nous pouvons même trouver (avec la même complexité) un plan temporel avec un nombre minimum d'instances d'actions.

Preuve : Ceci résulte presque directement du Théorème 1 et du fait que l'ensemble des contraintes d'autorisation, inhérentes, d'effets contradictoires et de causalité sont $STP^{\neq}$ [28]. Une instance de $STP^{\neq}$ peut être résolue en temps $O(n^3+k)$ et en espace $O(n^2+k)$ [13], où n est le nombre de variables et k le nombre d'inéquations (i.e. les contraintes de la forme $x_j - x_i \neq d$). Ici, les seules inéquations sont les contraintes d'effets contradictoires dont le nombre est au plus n^2, d'où $k=O(n^2)$. Par ailleurs, la calcul de SG et de A^r est clairement en $O(n^2)$.

La notion d'establisher-unique nous indique exactement quelles actions doivent appartenir à un plan temporel. Puis, (comme le montre la preuve du Théorème 1) les hypothèses de monotonie impliquent que nous n'avons besoin que d'une seule instance de chacune de ces actions. Il en résulte alors trivialement que nous résolvons la version optimale du problème de planification temporelle, dans lequel le but est de trouver un plan temporel avec un nombre minimal d'instances d'actions, en résolvant l'ensemble des contraintes d'autorisation, inhérentes, d'effets contradictoires et de causalité.

4 Détection de la Monotonie des Fluents

Une classe Π d'instances de problèmes NP-difficiles est généralement considérée comme traitable si elle satisfait deux conditions : (1) il existe un algorithme polynomial en temps pour résoudre Π, et (2) il existe un algorithme polynomial en temps pour reconnaître Π. Détecter si toutes les actions sont establisher-unique est clairement polynomial en temps. D'un autre côté, notre définition très générale de la monotonie des fluents implique que ce n'est pas le cas pour déterminer si des fluents sont monotones.

Théorème 3. Déterminer si un fluent d'un problème de planification temporelle <I,A,G> est monotone est PSPACE-difficile si le chevauchement d'instances d'une même action n'est pas permis dans les plans et EXPSPACE-complet si le chevauchement d'instances d'une même action est permis.

Preuve : Remarquons que si <I,A,G> n'a pas de solution, alors tous les fluents sont trivialement monotones par la Définition 7, puisqu'ils ne sont jamais établis ni détruits dans aucun plan. Il est suffisant d'ajouter deux nouveaux fluents buts f_1 et f_2 et deux nouvelles actions instantanées à A : a_1 qui ajoute simplement f_1 et a_2 qui a f_1 comme condition, ajoute f_2 et détruit f_1 (a_1 et a_2 étant indépendantes de tous les autres fluents) à tout problème <I,A,G> : f_1 est monotone si et seulement si le problème résultant n'a aucun plan temporel. Le théorème résulte alors du fait que tester l'existence d'un plan temporel pour un problème de planification temporelle <I,A,G> est PSPACE-difficile si le chevauchement d'instances d'une même action n'est pas permis dans les plans et EXPSPACE-complet si le chevauchement d'instances d'une même action est permis [29].

Nous pouvons néanmoins détecter la monotonie de certains fluents en temps polynomial. Il y a plusieurs façons de démontrer qu'un fluent est monotone. Dans cette section nous donnons quelques exemples qui donnent lieu à des algorithmes polynomiaux d'ordre faible. Etant donné

le Théorème 3, nous affirmons clairement que nous ne pouvons détecter tous les fluents monotones avec ces règles. L'ensemble des problèmes de planification temporelle dont les fluents peuvent être prouvés +monotone ou −monotone avec les règles données dans cette section, comme requis dans les conditions du Théorème 2, représentent une classe traitable, puisqu'elle peut à la fois être reconnue et résolue en temps polynomial.

Notre première règle reformule simplement le Lemme 2 de la section précédente.

Règle 1 : S'il n'existe pas d'actions $a, a' \in A^r$ telles que $f \in \text{Del}(a) \cap \text{Add}(a')$ alors f est à la fois +monotone et −monotone.

Il peut également être possible de montrer qu'un fluent est monotone à cause de ses liens avec d'autres fluents, déjà connus pour être monotones. Considérons un exemple simple d'un problème de planification temporelle avec deux fluents *item_in_shop*, *item_for_sale* et uniquement deux actions qui ont ces fluents comme condition ou effet : l'action DISPLAY_ITEM qui a pour condition *item_in_shop* et établit *item_for_sale*, et l'action SELL_ITEM qui a pour condition *item_for_sale* et détruit à la fois *item_for_sale* et *item_in_shop*. Pour simplifier, supposons que les deux actions sont instantanées. Le fluent *item_in_shop* est –monotone puisqu'il ne peut jamais être établi (il peut être présent ou pas dans l'état initial). Nous ne pouvons pas appliquer la Règle 1 pour déduire la monotonie de *item_for_sale*, puisqu'il peut à la fois être établi et détruit. Cependant, puisque *item_in_shop* est –monotone, il est clair que DISPLAY_ITEM ne peut être exécutée après SELL_ITEM. Il résulte alors que *item_for_sale* est –monotone. Nous formalisons cette idée dans les deux règles suivantes.

Règle 2 : Supposons que l'ensemble réduit d'actions A^r est establisher-unique relativement à l'ensemble des sous-buts SG (comme défini par la Définition 9) et soit a_f l'unique action qui établit le fluent $f \in$ SG. Si pour toute $a \in A^r$ telle que $f \in \text{Del}(a)$:

- ∃ un fluent −monotone $p \in \text{Del}(a) \cap \text{Cond}(a_f)$ tel que $\tau(a \rightarrow \neg p) - \tau(a \rightarrow \neg f) \leq \tau(p \rightarrow\mid a_f) - \tau(a_f \rightarrow f)$, ou
- ∃ un fluent −monotone $p \in \text{Del}(a) \cap \text{Add}(a_f)$ tel que $\tau(a \rightarrow \neg p) - \tau(a \rightarrow \neg f) \leq \tau(a_f \rightarrow p) - \tau(a_f \rightarrow f)$, ou
- ∃ un fluent +monotone $p \in \text{Add}(a) \cap \text{Del}(a_f)$ tel que $\tau(a \rightarrow p) - \tau(a \rightarrow \neg f) \leq \tau(a_f \rightarrow \neg p) - \tau(a_f \rightarrow f)$, ou
- ∃ un fluent +monotone $p \in (\text{Del}(a) \cap \text{Cond}(a_f))\backslash I$ tel que $\tau(p \rightarrow\mid a) - \tau(a \rightarrow \neg f) \leq \tau(a_f \rightarrow \neg p) - \tau(a_f \rightarrow f)$,

alors f est −monotone.

Règle 3 : Supposons que l'ensemble réduit d'actions A^r est establisher-unique relativement à l'ensemble des sous-buts SG et soit a_f l'unique action qui établit le fluent $f \in$ SG. Si pour toute $a \in A^r$ telle que $f \in \text{Del}(a)$:

- ∃ un fluent −monotone $p \in \text{Cond}(a) \cap \text{Del}(a_f)$ tel que $\tau(a_f \rightarrow \neg p) - \tau(a_f \rightarrow f) \leq \tau(p \rightarrow\mid a) - \tau(a \rightarrow \neg f)$, ou
- ∃ un fluent −monotone $p \in \text{Add}(a) \cap \text{Del}(a_f)$ tel que $\tau(a_f \rightarrow \neg p) - \tau(a_f \rightarrow f) \leq \tau(a \rightarrow p) - \tau(a \rightarrow \neg f)$, ou
- ∃ un fluent +monotone $p \in \text{Del}(a) \cap \text{Add}(a_f)$ tel que $\tau(a_f \rightarrow p) - \tau(a_f \rightarrow f) \leq \tau(a \rightarrow \neg p) - \tau(a \rightarrow \neg f)$, ou
- ∃ un fluent +monotone $p \in (\text{Del}(a) \cap \text{Cond}(a_f))\backslash I$ tel que $\tau(p \rightarrow\mid a_f) - \tau(a_f \rightarrow f) \leq \tau(a \rightarrow \neg p) - \tau(a \rightarrow \neg f)$,

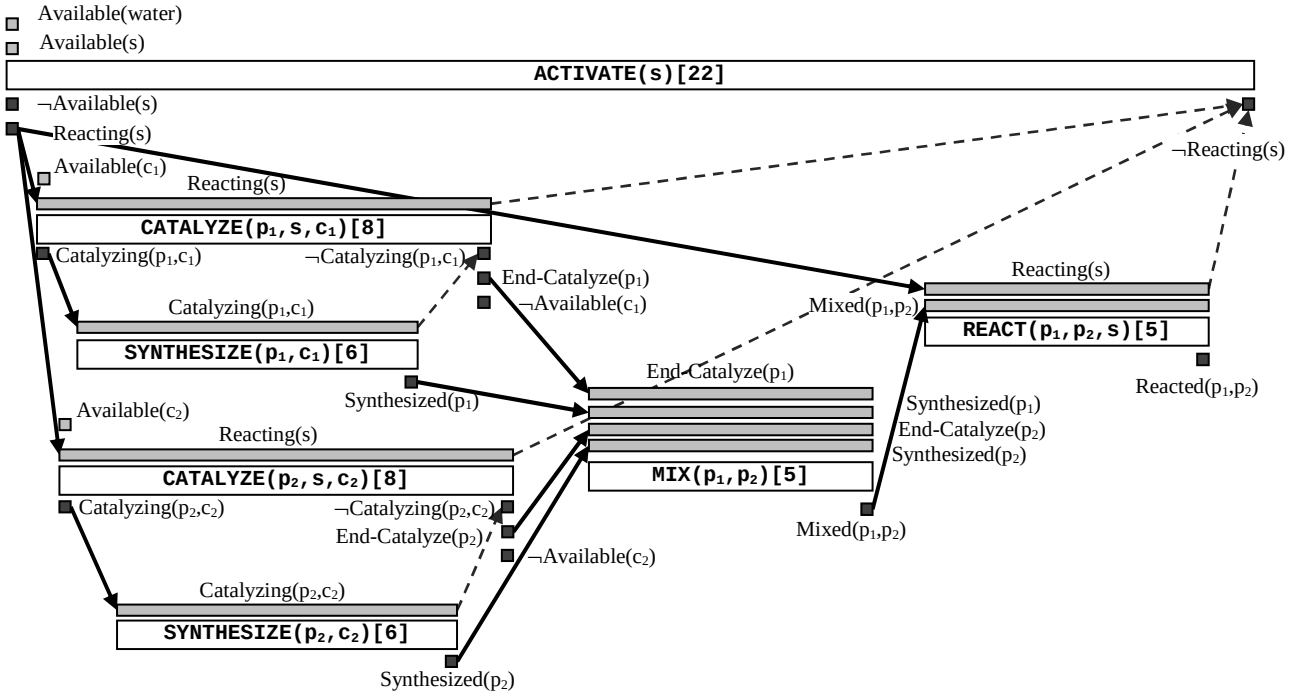
alors f est +monotone.

Proposition 2. Les règles 2 et 3 sont valides.

Preuve : Nous ne donnons que la preuve de validité de la Règle 2, puisque la preuve de la Règle 3 est entièrement similaire. Supposons que les prémisses de la Règle 2 soient vérifiées. Considérons un fluent arbitraire $f \in$ SG et soit a_f l'unique action qui établit le fluent $f \in$ SG. Supposons que $f \in \text{Del}(a)$ et qu'il existe un fluent −monotone $p \in \text{Del}(a) \cap \text{Cond}(a_f)$ tel que $\tau(a \rightarrow \neg p) - \tau(a \rightarrow \neg f) \leq \tau(p \rightarrow\mid a_f) - \tau(a_f \rightarrow f)$. Mais, puisque p est −monotone, nous savons que $\tau(p \rightarrow\mid a_f) < \tau(a \rightarrow \neg p)$. Il résulte directement que $\tau(a_f \rightarrow f) < \tau(a \rightarrow \neg f)$. Par un argument similaire pour les trois autres cas listés dans la Règle 2, nous pouvons déduire que pour toutes les actions $a \in A^r$ telles que $f \in \text{Del}(a)$, $\tau(a_f \rightarrow f) < \tau(a \rightarrow \neg f)$. Puisque a_f est l'unique action qui établit f, nous pouvons déduire que f ne peut jamais être établi après qu'il soit détruit et donc qu'il est −monotone.

Le théorème suivant résulte maintenant du fait que les Règles 1, 2 et 3 peuvent clairement être appliquées jusqu'à la convergence en temps polynomial.

Théorème 4. Soit Π la classe des problèmes de planification temporelle positifs <I,A,G> dans lesquels A est establisher-unique relativement à Cond(A) ∪ G, tous les fluents dans Cond(A) ∪ G sont monotones et tous les fluents dans I ∩ (Cond(A) ∪ G) sont −monotones, où la monotonie des fluents peut être détectée en appliquant les Règles 1, 2 et 3 jusqu'à convergence. Alors Π est traitable.



5 Un Exemple de Planification de Processus Chimique

Le domaine « Temporal Chemical Process » comporte différents types d'opérations sur des produits chimiques qui sont réalisées dans la production industrielle de composés. Il comporte par exemple un opérateur qui peut *activer* une source de matière première. Cette matière première peut ensuite être *catalysée* de deux manières pour *synthétiser* deux produits différents. Ces produits peuvent être *mélangés* et *réagir* en utilisant la matière première une nouvelle fois pour produire le composé souhaité. Ce processus est illustré par le plan temporel donné dans la figure qui suit. Nous représentons les actions non-instantanées par un rectangle. La durée d'une action est donnée entre crochets après son nom Les conditions sont données au-dessus d'une action, et ses effets en-dessous. L'état initial et le but du problème de planification correspondant sont :

`I={Available(water),Available(s),Available(c1), Available(c2)}`

`G={Reacted(p1,p2)}`

Etant donné le problème de planification temporelle <I,A,G>, où A est l'ensemble de toutes les actions du domaine « Temporal Chemical Process », l'ensemble des sous-buts SG et l'ensemble réduit d'actions A^r sont :

`SG={Reacted(p1,p2), Reacting(s), Mixed(p1,p2), Available(water), Available(s), End-Catalyze(p1), End-Catalyze(p2), Synthesized(p1), Synthesized(p2), Available(c1), Available(c2), Catalyzing(p1,c1), Catalyzing(p2,c2)}`

A^r=`{REACT(p1,p2,s), ACTIVATE(s), MIX(p1,p2), CATALYZE(p1,s,c1), SYNTHESIZE(p1,c1), CATALYZE(p2,s,c2), SYNTHESIZE(p2,c2)}`

Pour tout $i \neq j$ tels que $\{a_i, a_j\} \subset A^r$ nous avons $\text{Add}(a_i) \cap \text{Add}(a_j) \cap SG = \emptyset$. L'ensemble des actions A^r est donc establisher-unique relativement à SG. Nous pouvons remarquer immédiatement que le fluent `Available(water)` n'est jamais ajouté ou détruit, que les fluents `Reacted(p1,p2)`, `Mixed(p1,p2)`, `End-Catalyze(p1)`, `Synthesized(p1)`, `End-Catalyze(p2)`, `Synthesized(p2)` sont uniquement ajoutés, et les fluents `Available(s)`, `Available(c1)`, `Available(c2)` sont uniquement détruits. Ainsi, aucun de ces fluents n'est dans $\text{Add}(A^r) \cap \text{Del}(A^r)$, et par le Règle 1, ils sont –monotones. En utilisant la Règle 2, nous pouvons prouver que `Reacting(s)` est –monotone. a_f = `ACTIVATE(s)` est l'unique action qui établit le fluent f = `Reacting(s)` $\in$ SG. a = `ACTIVATE(s)` est également l'unique action qui détruit f = `Reacting(s)` et pour le fluent –monotone p = `Available(s)` $\in \text{Del}(a) \cap \text{Cond}(a_f)$, nous avons $\tau(a \rightarrow \neg p) - \tau(a \rightarrow \neg f) \leq \tau(p \rightarrow| a_f) - \tau(a_f \rightarrow f)$. Donc, par la Règle 2, `Reacting(s)` est –monotone. Par un argument similaire, en utilisant encore la Règle 2, nous pouvons prouver que `Catalyzing(p1,c1)` et `Catalyzing(p2,c2)` sont –monotones.

L'ensemble d'actions A^r est establisher-unique relativement à SG, chaque fluent dans $\text{Cond}(A^r) \cup G$ est monotone et chaque fluent dans $I \cap (\text{Cond}(A^r) \cup G)$ est –monotone relativement à $<I,A^r,G>$. En outre :
(1) $G \subseteq (I \backslash \text{Del}(A^r)) \cup \text{Add}(A^r)$
(2) $\text{Cond}(A^r) \subseteq I \cup \text{Add}(A^r)$

(3) tous les fluents $g \in G \cap Del(A^r) \cap Add(A^r)$ sont +monotone relativement à $<I,A^r,G>$ (trivialement, puisque cet ensemble est vide)
(4) il n'y a pas de contraintes d'effets contradictoires ni de +autorisation. L'ensemble des contraintes restantes a une solution sur l'ensemble des actions A^r.

Ainsi, par le Théorème 1, le problème <I,A,G> a un plan-solution, montré dans la figure (les contraintes de causalité sont représentées par les flèches en gras, et les contraintes de –autorisation par des flèches en pointillés), et par le Théorème 2, cette solution peut être trouvée en temps polynomial.

Beaucoup d'autres problèmes de planification temporelle issus de l'industrie chimique peuvent également être résolus en temps polynomial d'une manière similaire. Par exemple, l'acétylène est une matière première dérivée du carbure de calcium grace à l'eau. Ensuite, le chlorure de vinyle est un monomère produit à partir de l'acétylène et du chlorure d'hydrogène en utilisant du chlorure mercurique comme catalyseur. Le PVC est alors produit par polymérisation. D'autres exemples interviennent dans l'industrie pharmaceutique dans la production de médicaments (comme le paracétamol ou l'ibuprofène) et, en général, dans beaucoup de processus qui nécessitent la production et la combinaison de plusieurs molécules, sachant qu'il existe une façon unique de les obtenir (souvent imposée par des raisons industrielles, économiques ou écologiques).

6 Discussion

Les résultats obtenus ici peuvent également être appliqués à la planification non-temporelle puisque, par exemple, un problème de planification classique STRIPS peut être modélisé comme un problème de planification temporelle dans lequel toutes les actions sont instantanées. Il est intéressant de souligner que la classe traitable des problèmes de planification classique dans lesquels toutes les actions sont establisher-unique et ou tous les fluents sont détectables comme étant (à la fois + et –) monotones en appliquant seulement la Règle 1, est couverte par la classe traitable PA de [25].

Pour simplifier la présentation et pour demeurer dans le cadre de PDDL2.1, nous avons considéré que les contraintes inhérentes entre les instants correspondant à des événements à l'intérieur d'une même instance d'action sont toutes des contraintes d'intervalle. Nous pouvons cependant généraliser nos classes traitables pour autoriser des contraintes min-closed arbitraires puisque c'est la seule propriété requise pour les contraintes dans la preuve du Théorème 1. Un exemple d'une telle contrainte C(x,y) est une contrainte binaire d'intervalle avec des bornes variables : $y-x \in [f(x,y),g(x,y)]$, qui est min-closed à condition que f(x,y) soit une fonction monotone croissante de x et g(x,y) soit une fonction monotone décroissante de y. Un autre exemple de contrainte min-closed est la contrainte ternaire $(x+y)/2 \leq z$, qui peut par exemple être utilisée pour imposer qu'un effet ait lieu durant la dernière moitié de l'exécution d'une action.

Un aspect important de la planification temporelle qui est absent de la planification non-temporelle, est que certains problèmes de planification temporelle, appelés problèmes temporellement expressifs, nécessitent la concurrence des actions afin d'être résolus [8]. Un exemple typique de problème temporellement expressif est la cuisine : plusieurs ingrédients ou plats doivent être cuisinés simultanément afin d'être prêts au même moment. Dans les environnements industriels, la concurrence des actions est souvent utilisée pour conserver l'espace de stockage et les délais d'exécutions dans des limites données. [7] ont identifié une sous-classe des problèmes temporellement expressifs qu'ils ont appelés problèmes temporellement cycliques et qui nécessitent des ensembles d'actions cycliquement dépendantes afin d'être résolus. Un exemple simple de ce type de problème est le développement de deux parties d'un logiciel, écrites par deux sous-traitants différents, chacun devant connaître la spécification de l'autre programme afin de construire correctement leur interface. La classe traitable de problèmes de planification temporelle décrite dans le Théorème 4 contient à la fois des problèmes temporellement expressifs et des problèmes temporellement cycliques. Ceci résulte du fait que, comme illustré par l'exemple donné dans [7], il est possible de construire un exemple de problème temporellement cyclique qui soit establisher-unique et dans lequel aucun fluent n'est détruit par aucune action (et donc dans lequel, d'après le Lemme 2, tous les fluents sont à la fois + et – monotones). Le problème de planification du processus chimique donné dans la Section 5 est un autre exemple de problème qui est temporellement expressif puisque la concurrence des actions est requise dans toutes les solutions.

7 Conclusion

Nous avons présenté une classe de problèmes de planification temporelle qui peut être résolue en temps polynomial. Nous avons identifié un certain nombre d'applications possibles dans l'industrie chimique. De nouvelles recherches sont nécessaires pour trouver d'autres domaines d'applications possibles et, au niveau théorique, pour découvrir d'autres règles permettant de prouver la monotonie des fluents.

Références

[1] Bäckström C., Klein I. (1991) Parallel non-binary planning in polynomial time. Proceedings IJCAI'1991, pp. 268-273.

[2] Bäckström C., Nebel B. (1995) Complexity results for SAS+ planning). Computational Intelligence 11(4), pp. 625-655.

[3] Brafman R.I., Domshlak C. (2003) Structure and Complexity in Planning with Unary Operators. Journal of Artificial Intelligence Research 18, pp. 315-349.

[4] Brafman R.I., Domshlak C. (2006) Factored Planning: How, When, and When Not". Proceedings of the 21st National Conference on AI.

[5] Bylander T. (1994) The Computational Complexity of Propositional STRIPS Planning. Artificial Intelligence, 69(1-2), pp.165-204.

[6] Chen H., Giménez O. (2008) Causal Graphs and Structurally Restricted Planning. Proceedings of the 18th International Conference on Automated Planning and Scheduling, ICAPS'2008.

[7] Cooper M.C., Maris F., Régnier P. (2010) Solving temporally cyclic planning problems, International Symposium on Temporal Representation and Reasoning (TIME), p. 113-120.

[8] Cushing W., Kambhampati S., Mausam, Weld D.S. (2007) When is Temporal Planning Really Temporal? Proceedings of 20th International Joint Conference on Artificial Intelligence, IJCAI'2007, pp. 1852-1859.

[9] McDermott D. (1998) PDDL, The Planning Domain Definition Language. Technical Report, http://cs-www.cs.yale.edu/ homes/dvm/.

[10] Domshlak C., Dinitz Y. (2001) Multi-agent off-line coordination: Structure and complexity. Proceedings of 6th European Conference on Planning, ECP'2001.

[11] Erol K., Nau D.S., Subrahmanian V.S. (1995) Complexity, decidability and undecidability results for domain-independent planning. Artificial Intelligence, 76(1-2), pp.75-88.

[12] Fox M., Long D. (2003) PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains, Journal of Artificial Intelligence Research 20, pp. 61-124.

[13] Gerevini A., Cristani M. (1997) On Finding a Solution in Temporal Constraint Satisfaction Problems. Proceedings of 15th International Joint Conference on Artificial Intelligence, IJCAI'1997, pp. 1460-1465.

[14] Ghallab M., Nau D.S., Traverso P. (2004) Automated Planning: Theory and Practice, Morgan Kaufmann.

[15] O. Giménez, A. Jonsson (2008) The complexity of planning problems with simple causal graphs. Journal of AI Research 31, pp. 319-351.

[16] Haslum P. (2007) Reducing Accidental Complexity in Planning Problems. Proceedings of IJCAI'07, pp. 1898-1903.

[17] Haslum P. (2008) A New Approach To Tractable Planning. Proceedings of ICAPS'2008.

[18] Helmert M. (2003) Complexity results for standard benchmark domains in planning. Artificial Intelligence 143 (2), pp. 219-262.

[19] Helmert M. (2006) New Complexity Results for Classical Planning Benchmarks. Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS'2006, pp. 52-61.

[20] Hoffmann J. (2005) Where Ignoring Delete Lists Works, Local Search Topology in Planning Benchmarks. Journal of Artificial Intelligence Research 24, pp. 685-758.

[21] Jeavons P., Cooper M.C. (1995) Tractable constraints on ordered domains, Artificial Intelligence 79, pp. 327-339.

[22] Jonsson A. (2007) The Role of Macros in Tractable Planning Over Causal Graphs. Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'2007, pp. 1936-1941.

[23] Jonsson P., Bäckström C. (1994) Tractable planning with state variables by exploiting structural restrictions. Proceedings of AAAI'1994, pp. 998-1003.

[24] Jonsson P., Bäckström C. (1995) Incremental Planning. In New Directions in AI Planning: 3rd European Workshop on Planning, EWSP'1995, pp. 79-90.

[25] Jonsson P., Bäckström C. (1998) State-variable planning under structural restrictions: Algorithms and complexity. Artificial Intelligence, 100(1-2), pp. 125- 176.

[26] Katz M., Domshlak C. (2008) New Islands of Tractability of Cost-Optimal Planning. Journal of Artificial Intelligence Research, 32, pp. 203-288.

[27] Knoblock C.A. (1994) Automatically Generating Abstractions for Planning. Artificial Intelligence, 68(2), pp. 243-302.

[28] Koubarakis M. (1992) Dense Time and Temporal Constraints with ≠. Proceedings of 3rd International Conference on Principles of Knowledge Representation and Reasoning, KR'1992, pp. 24-35.

[29] Rintanen J. (2007) Complexity of concurrent temporal planning. Proceedings of the 17th International Conference on Automated Planning and Scheduling, ICAPS, pp. 280-287.

[30] Slaney J., Thiébaux S. (2001) Blocks World revisited. Artificial Intelligence 125, pp. 119-153.

[31] Vidal V., Geffner H. (2005) Solving Simple Planning Problems with More Inference and No Search. Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP'05, p. 682-696.

[32] Williams B.C., Nayak P. (1997) A reactive planner for a model-based executive. Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, pp. 1178-1185.

Substituabilité au voisinage pour le cadre WCSP

Christophe Lecoutre Olivier Roussel Djamel E. Dehani

Université Lille-Nord de France, Artois
CRIL - CNRS UMR 8188
F-62307 Lens
{lecoutre,roussel,dehani}@cril.fr

Résumé

WCSP est un problème d'optimisation pour lequel plusieurs formes de cohérences locales souples telles que, par exemple, la cohérence d'arc existentielle directionnelle (EDAC) et la cohérence d'arc virtuelle (VAC) ont été proposées durant ces dernières années. Dans cet article, nous adoptons une perspective différente en revisitant la propriété bien connue de la substituabilité (souple). Tout d'abord, nous précisons les relations existant entre la substituabilité de voisinage souple (SNS pour Soft Neighbourhood Substitutability) et une propriété appelée $pcost$ qui est basée sur le concept de surcoût de valeurs (par le biais de l'utilisation de paires de surcoût). Nous montrons que sous certaines hypothèses, $pcost$ est équivalent à SNS, mais que dans le cas général, elle est plus faible que SNS prouvée être coNP-difficile. Ensuite, nous montrons que SNS conserve la propriété VAC, mais pas la propriété EDAC. Enfin, nous introduisons un algorithme optimisé et nous montrons sur diverses séries d'instances WCSP l'intérêt pratique du maintien de $pcost$ avec AC*, FDAC ou EDAC, au cours de la recherche.

Abstract

WCSP is an optimization problem for which many forms of soft local (arc) consistencies such as existential directional arc consistency (EDAC) and virtual arc consistency (VAC) have been proposed these last years. In this paper, we adopt a different perspective by revisiting the well-known property of (soft) substitutability. First, we provide a clear picture of the relationships existing between soft neighborhood substitutability (SNS) and a tractable property called $pcost$ which is based on the concept of overcost of values (through the use of so-called cost pairs). We prove that under certain assumptions, $pcost$ is equivalent to SNS but weaker than SNS in the general case since we show that SNS is coNP-hard. We also show that SNS preserves the property VAC but not the property EDAC. Finally, we introduce an optimized algorithm and we show on various series of WCSP instances, the practical interest of maintaining $pcost$ together with AC*, FDAC or EDAC, during search.

1 Introduction

Le problème de satisfaction de contraintes valuées, VCSP pour Valued Constraint Satisfaction problem [23], est un cadre d'optimisation général utilisé avec succès pour manipuler le concept de contraintes souples dans de nombreuses applications en intelligence artificielle et en recherche opérationnelle. Une instance de problème VCSP est défini au moyen d'un ensemble de variables et d'un ensemble de fonctions de coût construites à partir d'une structure d'évaluation. Chaque fonction de coût détermine un degré de violation pour chaque instanciation possible d'un sous-ensemble de variables. Ces degrés (ou coûts) peuvent alors être combinés en utilisant l'opérateur $\oplus$ de la structure d'évaluation afin d'obtenir le coût global de toute instanciation complète. On peut globalement classer les différentes spécialisations du cadre VCSP selon les propriétés de l'opérateur $\oplus$: celles où $\oplus$ est idempotent (par exemple, min) et celles où $\oplus$ est (strictement) monotone (par exemple, $+$).

L'interchangeabilité est une propriété générale des réseaux de contraintes introduite par Freuder [13]. Deux valeurs a et b pour une variable x sont interchangeables si, pour toute solution I dans laquelle $x = b$, $I_{x=a}$ est également une solution, où $I_{x=a}$ désigne l'instanciation I dans laquelle a est assigné cette fois à x. L'interchangeabilité (complète) a été relaxée sous plusieurs formes telles que l'interchangeabilité de voisinage, la k-interchangeabilité, l'interchangeabilité partielle et la substituabilité. L'interchangeabilité et la substituabilité ont été

utilisées dans de nombreux contextes ; voir par exemple [2, 15, 6, 14, 1, 10, 22, 5, 18]. Une taxonomie partielle de ces deux propriétés peut être trouvée dans [16].

Une généralisation de la substituabilité pour les contraintes souples a été proposée dans [3] : une valeur a pour une variable x est susbtituable à une autre valeur b dans le domaine de x, si pour toute instanciation complète I impliquant (x, a), le coût de I est inférieur ou égal au coût de $I_{x=b}$. Il est particulièrement intéressant de noter que l'optimalité d'une instance est préservée lorsqu'on supprime une valeur pour laquelle une autre valeur est substituable. Calculer la substituabilité (complète) n'est pas traitable, mais la substituabilité de voisinage [3], qui est une forme limitée de la substituabilité où seules les contraintes impliquant une variable donnée sont considérées, peut être calculée en temps polynomial lorsque $\oplus$ est idempotent (à condition que l'arité des contraintes soit bornée). Cependant, lorsque $\oplus$ est monotone, comme c'est le cas pour la spécialisation VCSP appelée WCSP (Weighted CSP), aucun algorithme polynomial n'est connu.

Dans cet article, nous nous concentrons sur la substituabilité de voisinage souple (SNS) pour le cadre WCSP. Nous introduisons une propriété basée sur l'utilisation de paires de surcoût, appelée $pcost$, qui nous permet d'identifier efficacement des valeurs qui sont *souples-substituables*. Nous montrons que sous certaines hypothèses, $pcost$ est équivalent à SNS. Cependant, dans le cas général, lorsque k qui est la valeur qui représente le niveau des coûts interdits n'est pas ∞, $pcost$ est plus faible que SNS qui est par ailleurs démontré coNP-difficile. Nous étudions également les liaisons entre SNS et certaines propriétés connues de cohérence d'arc souple comme la cohérence d'arc existentielle directionnelle (EDAC) et la cohérence d'arc virtuelle (VAC). Nous démontrons que SNS préserve la propriété VAC, mais pas nécessairement la propriété EDAC. Enfin, nous développons un algorithme pour $pcost$ qui bénéficie d'une complexité en temps raisonnable, et nous montrons expérimentalement qu'il peut être associé à EDAC avec succès au cours de la recherche.

2 Contexte technique

Un *réseau de contraintes* (CN pour Constraint Network) P est un couple $(\mathscr{X}, \mathscr{C})$ où $\mathscr{X}$ est un ensemble fini de variables, noté par $vars(P)$, et $\mathscr{C}$ est un ensemble fini de contraintes. Chaque variable x possède un domaine (courant), noté $dom(x)$, qui est l'ensemble fini des valeurs qui peuvent être (couramment) affectées à x ; le domaine initial de x est noté $dom^{init}(x)$. d représente la taille du plus grand domaine. Chaque contrainte C_S implique un ensemble ordonné S de variables, appelés la *portée* de C_S, et représente une relation capturant l'ensemble des tuples autorisés pour les variables de S. Une contrainte *unaire* (resp., *binaire*) implique 1 (resp., 2) variable(s), et une contrainte *non-binaire* strictement plus que 2 variables. Une *instanciation* I d'un ensemble $X = \{x_1, \ldots, x_p\}$ de variables est un ensemble $\{(x_1, a_1), \ldots, (x_p, a_p)\}$ tel que $\forall i \in 1..p, a_i \in dom^{init}(x_i)$; X est noté $vars(I)$, et chaque a_i est noté par $I[x_i]$. Une instanciation I *sur* un CN P est une instanciation d'un ensemble $X \subseteq vars(P)$; elle est *complète* ssi $vars(I) = vars(P)$. I est *valide* sur P si et seulement si $\forall (x, a) \in I, a \in dom(x)$. I *couvre* une contrainte C_S ssi $S \subseteq vars(I)$. I *satisfait* une contrainte C_S avec $S = \{x_1, \ldots, x_r\}$ si et seulement si (i) I couvre C_S et (ii) le tuple $(I[x_1], \ldots, I[x_r]) \in C_S$. Une instanciation I sur un CN P est *localement cohérente* ssi (i) I est valide sur P et (ii) toutes les contraintes de P couvertes par I sont satisfaites par I. Une *solution* de P est une instanciation complète localement cohérente dans P.

Un *réseau de contraintes pondérées* (WCN pour Weighted CN) P est un triplet $(\mathscr{X}, \mathscr{C}, k)$ où $\mathscr{X}$ est un ensemble fini de n variables, $\mathscr{C}$ est un ensemble fini de e contraintes pondérées, également désigné par $cons(P)$, et $k > 0$ est un entier naturel ou $+\infty$. Chaque contrainte pondérée $c_S \in \mathscr{C}$ porte sur un ensemble ordonné S de variables (la portée), et est définie par une fonction de coût de $l(S)$ vers $\{0, \ldots, k\}$ où $l(S)$ est l'ensemble des instanciations possibles de S. Lorsque une instanciation a le coût k (noté aussi $\top$), elle est interdite. Sinon, elle est permise avec le coût correspondant (0, noté aussi par $\bot$, est entièrement satisfaisant). Afin de combiner les coûts, nous avons besoin de l'opérateur binaire $\oplus$ défini comme suit :

$$\forall a, b \in \{0, \ldots, k\}, a \oplus b = min(k, a + b)$$

Pour toute instanciation I et tout ensemble de variables X, soit $I_{\downarrow X} = \{(x, a) \mid (x, a) \in I \wedge x \in X\}$ la projection de I sur X. Nous désignons par $I_{x=a}$ l'instanciation $I_{\downarrow X \setminus \{x\}} \cup \{(x, a)\}$, qui est obtenue à partir de l'instanciation I soit par la substitution de la valeur affectée à x dans I par a, soit par l'extension de I avec (x, a). L'ensemble des contraintes voisines de x est désigné par $\Gamma(x) = \{c_S \in cons(P) \mid x \in S\}$. Quand $\Gamma(x)$ ne contient pas deux contraintes partageant au moins deux variables, on dit que $\Gamma(x)$ est *séparable*. Si c_S est une contrainte (pondérée) et I est une instanciation d'un ensemble $X \supseteq S$, alors $c_S(I)$ sera considéré comme égal à $c_S(I')$ où I' est la restriction de I aux variables de S (en d'autres termes, les projections seront implicites). Si C est un ensemble de contraintes, alors $vars(C) = \cup_{c_S \in C} S$ est l'ensemble des variables impliquées dans C ; si I est une instanciation telle que $vars(C) \subseteq vars(I)$, alors $cost_C(I) = \bigoplus_{c_S \in C} c_S(I)$ est le coût de I obtenu en considérant toutes les contraintes de C. Pour un WCN P et une instanciation complète I de P, le coût de I est alors $cost_{cons(P)}(I)$ qui sera simplifié en $cost(I)$. La tâche usuelle (NP-difficile) du problème de satisfaction de contraintes pondérées (WCSP) est de trouver, pour un WCN donné, une instanciation complète dont le coût est minimal. CSP peut être considéré comme une spécialisation de WCSP (lorsque seulement les coûts 0 et

k sont utilisés) et WCSP [20] peut être considéré comme une spécialisation du cadre générique de contraintes valuées [4].

De nombreuses formes de cohérence d'arc souple ont été proposées au cours des dernières années (voir par ex. [9]). Nous allons brièvement les introduire dans le contexte des WCNs binaires. Sans perte de généralité, on supposera l'existence d'une contrainte d'arité zéro notée $c_\emptyset$ (une constante) ainsi que la présence d'une contrainte unaire c_x pour chaque variable x. Une variable x est noeud-cohérente (NC*) si et seulement si $\forall a \in dom(x), c_\emptyset \oplus c_x(a) < k$ et $\exists b \in dom(x) \mid c_x(b) = 0$. Une variable x est arc-cohérente (AC*) si et seulement si x est NC* et $\forall a \in dom(x), \forall c_{xy} \in cons(P), \exists b \in dom(y) \mid c_{xy}(a,b) = 0$; b est appelé un support simple de a. Un WCN est AC^* ssi chacune de ses variables est AC^* [19, 20]. Un WCN est arc-cohérent directionnel complet (FDAC) [7] par rapport à un ordre total $<$ sur les variables ssi il est AC^* et $\forall c_{xy} \mid x < y, \forall a \in dom(x), \exists b \in dom(y) \mid c_{xy}(a,b) = c_y(b) = 0$; b est appelé un support complet de a. Un WCN est arc-cohérent existentiel (EAC) [12] ssi il est NC* et $\forall x \in vars(P), \exists a \in dom(x) \mid c_x(a) = 0 \wedge \forall c_{xy}, \exists b \in dom(y) \mid c_{xy}(a,b) = c_y(b) = 0$ (a est appelé le support existentiel de x) . Un WCN est arc-cohérent directionnel existentiel (EDAC) par rapport à un ordre $<$ sur les variables ssi il est EAC et FDAC par rapport à $<$. On notera $\phi(P)$ l'établissement de la propriété ϕ (par exemple AC ou EDAC) sur le (W)CN P. Pour tout WCN P, nous pouvons construire un CN noté $Bool(P)$ qui est obtenu en transformant chaque contrainte souple en une contrainte (dure) où seuls les tuples avec un coût nul sont autorisés. P est virtuel arc-cohérent (VAC) [8] ssi $AC(Bool(P)) \neq \bot$. Un WCN est optimal arc-cohérent souple (OSAC) ssi aucune transformation EPT (voir [9]) qui peut lui être appliquée ne permet d'augmenter $c_\emptyset$. OSAC est plus fort que VAC, qui lui-même est plus fort que EDAC, lorsqu'on compare les valeurs de $c_\emptyset$.

3 La substituabilité souple

Dans cette section, nous introduisons la substituabilité souple (au voisinage) [13, 3]. À partir de maintenant, nous considérons un (W)CN P donné.

Définition 1 *Soient $x \in vars(P)$ et $\{a,b\} \subseteq dom(x)$. (x,a) est* souple-substituable *à (x,b) dans P ssi pour toute instanciation complète I de P, $cost(I_{x=a}) \leq cost(I_{x=b})$.*

Lorsque (x,a) est souple-substituable à (x,b), b peut être supprimé de $dom(x)$ sans changer le coût optimum de P. En effet, si elles existent, les solutions optimales possibles de P avec (x,b) sont perdues, mais il est garanti qu'il reste au moins une solution optimale avec (x,a).

Parce qu'elle implique de traiter toutes les instanciations complètes, l'identification des valeurs souples-substituables est inutilisable en pratique. Cependant, il y a une forme de substituabilité locale, appelé substituabilité au voisinage [13, 3] qui peut être utile. Pour le cadre WCSP, elle est définie comme suit :

Définition 2 *Soient $x \in vars(P)$ et $\{a,b\} \subseteq dom(x)$, (x,a) est souple-substituable à (x,b)* au voisinage *dans P ssi pour chaque instanciation complète I de P, $cost_{\Gamma(x)}(I_{x=a}) \leq cost_{\Gamma(x)}(I_{x=b})$.*

On dira que (x,b) est SNS-éliminable (dans P) quand il existe une valeur (x,a) souple-substituable à (x,b) au voisinage. La substituabilité souple au voisinage implique la substituabilité souple (complète) (mais l'inverse n'est pas vrai). Il est particulièrement intéressant de noter que la substituabilité souple au voisinage permet une compensation de coûts entre contraintes souples. Une telle compensation est rendue possible par la présence de tous les coûts intermédiaires dans la structure d'évaluation, c'est-à-dire, les coûts différents de 0 et de k Par exemple, considérons un WCN composé de trois variables x, y et z avec $dom(x) = dom(y) = dom(z) = \{a,b\}$ et deux contraintes pondérées c_{xy} et c_{xz} telles que $c_{xy}(a,a) = c_{xz}(b,a) = c_{xz}(b,b) = 1$; tous les autres coûts étant à 0. Une illustration de ce WCN est donnée par la figure 1 ; les contraintes binaires sont représentées avec des arêtes étiquetées, et les coûts nuls ne sont pas représentés. Notons que (x,a) est souple-substituable à (x,b) au voisinage grâce à la compensation des coûts.

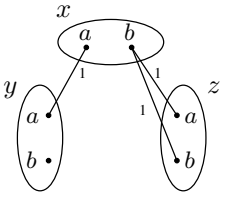


FIGURE 1 – (x,a) est souple-substituable à (x,b) au voisinage

La définition 2 nécessite de considérer chaque instanciation de $vars(\Gamma(x))$, ce qui reste très coûteux. La considération de chaque contrainte prise individuellement permettrait de réduire ce coût, mais dans ce cas, il est nécessaire de pouvoir identifier les compensations de coûts entre les différentes contraintes. Une façon simple de le faire est de calculer une somme de surcoûts minimaux, c'est à dire, une somme de différences de coût minimales sur toutes les contraintes : $ocost(x : a \rightarrow b) = \sum_{c_S \in \Gamma(x)} min_{I \in l(S)} \{c_S(I_{x=b}) - c_S(I_{x=a})\}$, tel que mentionné dans [17, 11]. Malheureusement, cette soustraction de coûts pose des problèmes subtils quand $k \neq \infty$. Ceci est illustré ci-dessous.

Exemple 1 Considérons les deux familles de contraintes $C_i = \{c_i \mid i \in 1..n\}$ et $C'_i = \{c'_i \mid i \in 1..n'\}$ définis par :

x	y_i	c_i
a	c	0
b	c	1

x	z_i	c'_i
a	d	1
b	d	0

Quand $n = k$ et $n' = k + 1$, (x, a) et (x, b) sont interchangeable parce que les deux valeurs sont interdites. Cependant, $\forall c_i \in C_i, cost_{c_i}(I_{x=b}) - cost_{c_i}(I_{x=a}) = 1$ et $\forall c'_i \in C'_i, cost_{c'_i}(I_{x=b}) - cost_{c'_i}(I_{x=a}) = -1$. En considérant la somme des différences sur les contraintes de $\Gamma(x) = C_i \cup C'_i$, la valeur résultante est -1 ce qui indiquerait que b a globalement un coût inférieur à celui a, ce qui est faux puisque les deux valeurs a et b ont un coût de k. Pour identifier correctement ce cas de substituabilité souple lorsque $k \neq \infty$, il est obligatoire d'utiliser un opérateur non commutatif, qui nous empêche d'utiliser l'opérateur $-$ usuel.

4 Calcul de la substituabilité souple

Nous allons maintenant nous concentrer sur la substituabilité souple au voisinage, et plus précisément sur la complexité de l'identification des valeurs SNS-éliminables. Nous commençons par discuter de certains travaux connexes à notre démarche. Pour le cadre général VCSP, des algorithmes efficaces pour la substituabilité souple au voisinage existent [3] lorsque l'opérateur d'agrégation de la structure d'évaluation VCSP est idempotent. Pour le cadre FCSP (Fuzzy CSP), la notion de substituabilité floue au voisinage est proposée dans [7] : il est montré que les valeurs floues-substituables au voisinage peuvent être efficacement identifiées lorsque l'opérateur d'agrégation de la structure d'évaluation FCSP est strictement monotone ou lorsque il s'agit de l'opérateur max. Plus récemment, la possibilité de calculer des formes de dominance plus faibles que la substituabilité souple au voisinage a été proposé dans [17]. Cependant, aucune étude précise qualitative n'a été menée pour le cadre WCSP. C'est ce que nous proposons de faire maintenant.

Tout d'abord, nous introduisons les paires de surcoût sur lesquelles notre mécanisme de calcul se base (ceci peut aussi être relié à ce qui a été proposé dans [7] pour le cadre FCSP). En effet, une manière de contourner les problèmes de soustraction mentionnés ci-dessus est d'utiliser seulement l'addition sur des paires de surcoût. Cette méthode est analogue à la construction d'entiers comme classes d'équivalence de paires ordonnées de nombres naturels où une paire (β, α) représente l'entier $\beta - \alpha$. Nous définissons $+$ (addition) sur les paires de surcoût par $(\beta, \alpha) + (\beta', \alpha') = (\beta + \beta', \alpha + \alpha')$ (ceci est le $+$ usuel et non $\oplus$) et la comparaison des paires de surcoût avec 0 par $(\beta, \alpha) \geq 0 \Leftrightarrow \beta \geq \alpha$. Les paires sont ordonnées par la relation $\leq$ définie par $(\beta, \alpha) \leq (\beta', \alpha') \Leftrightarrow (\beta - \alpha < \beta' - \alpha') \vee (\beta - \alpha = \beta' - \alpha' \wedge \alpha < \alpha')$. En un sens, la paire (β, α) porte deux informations : la différence $\beta - \alpha$ mais aussi $min(\beta, \alpha)$ qui est perdu lorsqu'on utilise une simple soustraction.

Définition 3 *Soient $x \in vars(P)$ et $\{a, b\} \subseteq dom(x)$,*
- *la* paire de surcoût *de (x, b) vis à vis de (x, a) dans $c_S \in \Gamma(x)$ est définie par $pcost(c_S, x : a \to b) = min_{I \in l(S)}\{(c_S(I_{x=b}), c_S(I_{x=a}))\}$;*
- *la* paire de surcoût *de (x, b) vis à vis de (x, a) dans P est définie par $pcost(x : a \to b) = \sum_{c_S \in \Gamma(x)} pcost(c_S, x : a \to b)$.*

Proposition 1 *Soient $x \in vars(P)$ et $\{a, b\} \subseteq dom(x)$. Si $pcost(x : a \to b) \geq 0$ alors (x, a) est souple-substituable à (x, b) au voisinage dans P.*

Preuve. Pour une contrainte c_S, soit I^{c_s} l'instanciation de $S - \{x\}$ qui donne la paire de surcoût minimale dans $min_{I \in l(S)}\{(c_S(I_{x=b}), c_S(I_{x=a}))\}$. Par définition, $pcost(c_S, x : a \to b) = (c_S(I^{c_S}_{x=b}), c_S(I^{c_S}_{x=a}))$. Par définition de min sur les paires de surcoût, nous avons $\forall I, \forall c_S \in \Gamma(x)$, $(c_S(I_{x=b}), c_S(I_{x=a})) \geq (c_S(I^{c_S}_{x=b}), c_S(I^{c_S}_{x=a}))$. Par sommation, nous obtenons $\forall I$, $\sum_{c_S \in \Gamma(x)}(c_S(I_{x=b}), c_S(I_{x=a})) \geq \sum_{c_S \in \Gamma(x)}(c_S(I^{c_S}_{x=b}), c_S(I^{c_S}_{x=a}))$. Par hypothèse, $pcost(x : a \to b) \geq 0$, donc nous avons $\sum_{c_S \in \Gamma(x)}(c_S(I^{c_S}_{x=b}), c_S(I^{c_S}_{x=a})) \geq 0$, et donc $\forall I, \sum_{c_S \in \Gamma(x)}(c_S(I_{x=b}), c_S(I_{x=a})) \geq 0$. Par définitions de $+$ et $\leq$ sur les paires de surcoût, nous pouvons tirer $\forall I, \sum_{c_S \in \Gamma(x)} c_S(I_{x=b}) \geq \sum_{c_S \in \Gamma(x)} c_S(I_{x=a})$ ce qui implique $\forall I, min(k, \sum_{c_S \in \Gamma(x)} c_S(I_{x=b})) \geq min(k, \sum_{c_S \in \Gamma(x)} c_S(I_{x=a}))$. Puisque $\forall a_i \in \{0, \ldots, k\}$, $a_1 \oplus \ldots \oplus a_n = min(k, a_1 + \ldots a_n\}$, nous pouvons conclure que $\forall I, \bigoplus_{c_S \in \Gamma(x)} c_S(I_{x=b})) \geq \bigoplus_{c_S \in \Gamma(x)} c_S(I_{x=a}))$. Ainsi, (x, a) est souple-substituable à (x, b) au voisinage dans P. □

La réciproque de la proposition 1 n'est pas vraie dans le cas général. Un premier cas où elle est fausse apparaît lorsque les portées de deux contraintes non binaires ont une intersection de plus d'une variable (voisinage non séparable). Dans cette situation, les contraintes ne peuvent pas être considérées individuellement.

Exemple 2 Soient quatre variables x, y, z et t telles que $dom(x) = \{a, b\}, dom(y) = \{c, d\}, dom(z) = dom(t) = \{e\}$, et deux contraintes ternaires c_{xyz}, c_{xyt} définies par la table de coûts suivante :

x	y	z	t	c_{xyz}	c_{xyt}
a	c	e	e	1	0
b	c	e	e	0	1
a	d	e	e	0	1
b	d	e	e	1	0

Il est assez facile de constater que (x, a) est souple-substituable à (x, b) au voisinage alors que $pcost(c_{xyz}, x, a \to b) = pcost(c_{xyt}, x, a \to b) = (0, 1)$ et, par conséquent $pcost(x, a \to b) = (0, 2) \not\geq 0$.

Ainsi, une première condition pour que la réciproque de la proposition 1 tienne est que $\Gamma(x)$ soit séparable (ce qui est le cas des réseaux binaires normalisés). Un autre cas de figure où la réciproque de la proposition 1 est fausse est quand $k \neq \infty$. Considérant le WCN de l'exemple 1 avec $n = k$ et $n' = k + 1$, nous pouvons observer que $pcost(x : a \to b) = (n, n') = (k, k + 1) \not\geq 0$. Cependant, (x, a) et (x, b) sont tous deux interdits et par conséquent (x, a) est souple-substituable à (x, b) (et inversement). Dans cet exemple, il pourrait sembler une bonne idée d'utiliser $\oplus$ au lieu de $+$ pour l'addition de paires. Cependant, l'exemple 3 montre que cela conduirait à l'identification erronée de valeurs substituables.

Exemple 3 Considérons la contrainte unaire c_x et la famille de contraintes binaires $C_i = \{c_i \mid i \in 1..n\}$ définie par :

x	y_i	c_i
a	a	2
a	b	0
b	a	1
b	b	0

x	c_x
a	1
b	0

Clairement, (x, a) est non substituable à (x, b). Posons $n = k$. Avec la somme des paires définie par $+$, $pcost(x, a \to b) = (n, 2n+1) \not\geq 0$. Si la somme de paires est définie par $\oplus$, nous obtiendrions $(k, k) \geq 0$.

Heureusement, il y a des situations où, même lorsque $k \neq \infty$, l'utilisation des paires de surcoût nous permet d'identifier précisément l'ensemble des valeurs souples-substituables au voisinage.

Proposition 2 *Soient $x \in vars(P)$ et $\{a, b\} \subseteq dom(x)$ tel que $\Gamma(x)$ est séparable et $pcost(x, a \to b) = (\beta, \alpha)$ avec $\alpha < k$. Si (x, a) est souple-substituable à (x, b) au voisinage dans P alors $pcost(x : a \to b) \geq 0$.*

Preuve. Puisque par hypothèse $\Gamma(x)$ est séparable, nous pouvons définir l'instanciation I^{min} dans $vars(\Gamma(x)) \setminus \{x\}$ comme l'union pour chaque contrainte $c_s \in \Gamma(x)$ de l'instanciation I^{c_S} définie dans la preuve de la proposition 1. I^{min} est tel que $pcost(c_S, x : a \to b) = (c_S(I^{min}_{x=b}), c_S(I^{min}_{x=a}))$.

Par hypothèse, $\forall I, cost_{\Gamma(x)}(I_{x=b}) \geq cost_{\Gamma(x)}(I_{x=a})$, ce qui peut être réécrit sous la forme $\forall I, \bigoplus_{c_S \in \Gamma(x)} c_S(I_{x=b}) \geq \bigoplus_{c_S \in \Gamma(x)} c_S(I_{x=a})$. En particulier, cela est vrai pour $I = I^{min}$. Par conséquent, $\bigoplus_{c_S \in \Gamma(x)} c_S(I^{min}_{x=b}) \geq \bigoplus_{c_S \in \Gamma(x)} c_S(I^{min}_{x=a})$ qui peut être réécrit sous la forme $min(k, \beta) \geq min(k, \alpha)$ où $\beta = \sum_{c_S \in \Gamma(x)} c_S(I^{min}_{x=b}))$ et $\alpha = \sum_{c_S \in \Gamma(x)} c_S(I^{min}_{x=a})$. Par définition, $pcost(x : a \to b) = (\beta, \alpha)$ et donc $pcost(x : a \to b) \geq 0$ ssi $\beta \geq \alpha$. Maintenant, si $\alpha < k$, $min(k, \alpha) = \alpha$ et $min(k, \beta) \geq min(k, \alpha) \Rightarrow \beta \geq \alpha \Rightarrow pcost(x : a \to b) \geq 0$ (ceci est vrai pour $\beta < k$ et $\beta \geq k$). Notons que lorsque $\alpha \geq k$, $min(k, \beta) \geq min(k, \alpha) \not\Rightarrow \beta \geq \alpha$; un contre-exemple étant $\beta = k$ et $\alpha = k + 1$. □

Corollaire 1 *Soient $x \in vars(P)$ et $\{a, b\} \subseteq dom(x)$ tel que $\Gamma(x)$ est séparable et $pcost(x : a \to b) = (\beta, \alpha)$ avec $\alpha < k$. Si $(\beta, \alpha) < 0$ alors (x, a) n'est pas souple-substituable à (x, b) au voisinage dans P.*

Quand $pcost(x : a \to b) = (\beta, \alpha)$ avec $\alpha \geq k$, décider si (x, a) est souple-substituable à (x, b) au voisinage est beaucoup plus difficile. En effet, ce problème est coNP difficile. Pour prouver cela, nous introduisons le problème de double coût à choix multiple.

Le problème de choix multiple à double coût (MCDCP) Étant donnés m ensembles E_1, E_2, ..., E_m d'objets tels que chaque objet $o_j \in E_i$ a une valeur de coût principale $r_{ij} \in \mathbb{Z}^+$ ainsi qu'une valeur de coût secondaire $s_{ij} \in \mathbb{Z}^+$. Étant donné un coût maximal $C \in \mathbb{Z}^+$, le problème MCDCP consiste à décider si il est possible de choisir un objet dans chaque ensemble de telle sorte que la somme des coûts principaux de ces objets sélectionnés ne dépasse pas C et ne dépasse pas également la somme des coûts secondaires. Ce problème peut être formulé comme suit :

$$\begin{array}{l} \sum_{i=1}^{m} \sum_{j \in E_i} r_{ij} x_{ij} \leq C, \\ \sum_{i=1}^{m} \sum_{j \in E_i} r_{ij} x_{ij} \leq \sum_{i=1}^{m} \sum_{j \in E_i} s_{ij} x_{ij}, \\ \sum_{j \in E_i} x_{ij} = 1, i = 1, \ldots, m, \\ x_{ij} \in \{0, 1\}, i = 1, \ldots, m, j \in E_i. \end{array}$$

Proposition 3 *Le problème de choix multiple à double coût est NP-complet.*

Preuve. L'adhésion à NP est immédiat. Pour la NP-difficulté, nous réduisons le problème de sac à dos à choix multiple (MCKP pour Multiple-Choice Knapsack Problem) au problème de choix multiple à double coût. MCKP est connu pour être NP-difficile (par exemple, voir [21]). Pour MCKP, nous avons aussi m ensembles, et un profit $p_{ij} \in \mathbb{Z}^+$ est associé à chaque objet ainsi qu'un poids $w_{ij} \in \mathbb{Z}^+$. Étant donné un bénéfice minimal $P \in \mathbb{Z}^+$ et un poids maximal $W \in \mathbb{Z}^+$, le problème de décision MCKP est formulé comme suit :

$$\begin{array}{l} \sum_{i=1}^{m} \sum_{j \in E_i} w_{ij} x_{ij} \leq W, \\ \sum_{i=1}^{m} \sum_{j \in E_i} p_{ij} x_{ij} \geq P, \\ \sum_{j \in E_i} x_{ij} = 1, i = 1, \ldots, m, \\ x_{ij} \in \{0, 1\}, i = 1, \ldots, m, j \in E_i. \end{array}$$

Pour coder une instance MCKP en une instance MCDCP, nous conservons la même structure (ensembles) et définissons :

$C = qW - mP$,
$r_{ij} = qw_{ij} - P, i = 1, \ldots, m, j \in E_i$,
$s_{ij} = mp_{ij} + r_{ij} - P, i = 1, \ldots, m, j \in E_i$,

avec $q = 2mP$. Avec cette valeur choisie pour q, on peut montrer que toutes les valeurs C, r_{ij} et s_{ij} appartiennent à $\mathbb{Z}^+$. La première équation MCDCP peut être transformée comme suit : $\sum_{i=1}^{m} \sum_{j \in E_i} r_{ij} x_{ij} \leq C$
$\Rightarrow \sum_{i=1}^{m} \sum_{j \in E_i} (qw_{ij} - P) x_{ij} \leq qW - mP$
$\Rightarrow \sum_{i=1}^{m} \sum_{j \in E_i} qw_{ij} x_{ij} - mP \leq qW - mP$ car exactement m variables x_{ij} sont assignées à 1,
$\Rightarrow \sum_{i=1}^{m} \sum_{j \in E_i} w_{ij} x_{ij} \leq W$.
La seconde équation MCDCP peut être transformée comme suit : $\sum_{i=1}^{m} \sum_{j \in E_i} r_{ij} x_{ij} \leq \sum_{i=1}^{m} \sum_{j \in E_i} s_{ij} x_{ij}$,
$\Rightarrow 0 \leq \sum_{i=1}^{m} \sum_{j \in E_i} (mp_{ij} - P) x_{ij}$ en simplifiant r_{ij} des deux côtés,
$\Rightarrow 0 \leq \sum_{i=1}^{m} \sum_{j \in E_i} mp_{ij} x_{ij} - mP$ car exactement m variables x_{ij} sont assignées à 1,
$\Rightarrow P \leq \sum_{i=1}^{m} \sum_{j \in E_i} p_{ij} x_{ij}$. □

Proposition 4 *Décider si une valeur est souple-substituable au voisinage à une autre dans un WCN $(\mathscr{X}, \mathscr{C}, k)$ où $k \neq \infty$ est coNP-difficile.*

Preuve. Toute instance MCDCP peut être réduit au problème de décider si une valeur (x, a) n'est pas souple-substituable à une autre valeur (x, b) au voisinage dans un WCN P. A partir de l'instance MCDCP, nous construisons le WCN P comme suit :

- $vars(P)$ contient une variable x tel que $dom(x) = \{a, b\}$ et une variable y_i pour chaque ensemble E_i ; le domaine de y_i contient les objects o_{i1}, o_{i2}, ... de E_i.
- $cons(P)$ contient exactement m contraintes binaires souple c_{xy_i} : nous avons $c_{xy_i}(\{(x, a), (y_i, o_{ij})\}) = s_{ij}$ et $c_{xy_i}(\{(x, b), (y_i, o_{ij})\}) = r_{ij}$.
- k est fixé à C.

Déterminer si (x, a) n'est pas souple-substituable à (x, b) au voisinage dans P est équivalent à trouver une instanciation I de $vars(\Gamma(x))$ telle que $cost_{\Gamma(x)}(I_{x=a}) > cost_{\Gamma(x)}(I_{x=b})$ qui est équivalent à $\sum_{c_S \in \Gamma(x)} c_s(I_{x=b}) < k \wedge \sum_{c_S \in \Gamma(x)} c_s(I_{x=a}) > \sum_{c_S \in \Gamma(x)} c_s(I_{x=b})$. La première (resp. seconde) condition encode la première (resp. seconde) inégalité de l'instance MCDCP. Comme les variables x_{ij} de l'instance MCDCP correspondent à l'affectation des variables y_i dans le WCN ($x_{ij} = 1 \Leftrightarrow y_i = o_{ij}$), la troisième équation de l'instance MCDCP est directement prise en compte dans le WCN. □

En pratique, il est plus facile de manipuler des différences de coûts en utilisant $ocost(x : a \rightarrow b) = \sum_{c_S \in \Gamma(x)} min_{I \in l(S)} \{c_S(I_{x=b}) - c_S(I_{x=a})\}$. En un sens, $pcost$ est plus précis que $ocost$: quand $pcost(x : a \rightarrow b) = (\beta, \alpha)$ avec $\alpha < k$, (x, a) souple-substituable à (x, b) au voisinage est équivalent à $\beta \geq \alpha$, et quand $\alpha \geq k$, aucune conclusion ne peut être donnée. Avec $ocost$, l'information α est perdue. Par conséquent, lorsque $ocost(x : a \rightarrow b) < 0$, nous ne pouvons pas conclure avec certitude que (x, a) n'est pas souple-substituable à (x, b) au voisinage. Dans la pratique, cela fait peu de différence (tout du moins, si on considère nos développements algorithmiques actuels), ce qui est la raison pour laquelle les expériences dans le présent document ont été réalisées avec $ocost$.

5 Liaisons avec la cohérence d'arc souple

Après l'introduction de la fermeture par substituabilité souple au voisinage, cette section présente quelques résultats connectant la substituabilité souple au voisinage avec diverses formes de cohérence d'arc souple.

Définition 4 *La fermeture par substituabilité souple au voisinage (ou SNS-closure) d'un WCN P, noté $SNS(P)$, est tout WCN obtenu après l'élimination itérative des valeurs SNS-éliminables jusqu'à ce qu'un point fixe soit atteint.*

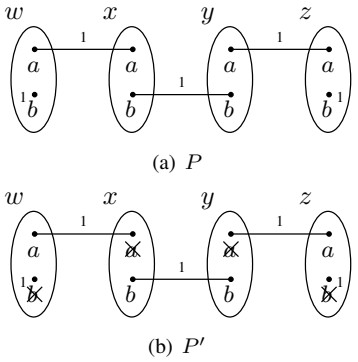
Puisque cette opération n'est pas confluente, $SNS(P)$ n'est pas unique. Quand nous utilisons l'approche $pcost$ pour identifier des valeurs SNS-éliminables, on notera $PSNS(P)$.

Proposition 5 *Soit P un WCN EDAC-cohérent. $SNS(P)$ n'est pas nécessairement EDAC-cohérent.*

Preuve. Considérons le WCN P représenté par la figure 2(a). Notons que P est EDAC-cohérent par rapport à l'ordre $w > x > z > y$, et que (w, a) et (z, a) sont respectivement souples-substituables à (w, b) et (z, b) au voisinage, puisque $pcost(w, a \rightarrow b) = (0, 0)$ et $pcost(z, a \rightarrow b) = (0, 0)$. Il existe une SNS-closure unique de P, $P' = SNS(P)$, qui est représentée par la figure 2(b). Il est clair que P', n'est pas EDAC-cohérent puisque (x, b) et (y, b) n'ont pas de support sur c_{xy}. □

Lemma 1 *Soient P un WCN VAC-cohérent, $x \in S$ et $\{a, b\} \subseteq dom(x)$ tel que (x, b) est AC-cohérent dans $Bool(P)$. Si $pcost(x : a \rightarrow b) \geq 0$ dans P alors (x, a) est substituable (au sens CSP [13]) à (x, b) au voisinage dans $Bool(P)$.*

Preuve. Nous supposons que P est VAC-cohérent et (x, b) est AC-cohérent dans $Bool(P)$. Puisque (x, b) est AC-cohérent dans $Bool(P)$, nous savons que pour chaque contrainte $c_S \in \Gamma(x)$, il existe une instanciation I de S telle que $I[x] = b$ et $c_S(I) = 0$ (par construction de $Bool(P)$). Cela veut dire que pour chaque contrainte



(a) P

(b) P'

FIGURE 2 – EDAC versus SNS

$c_S \in \Gamma(x)$, $pcost(c_S, x : a \to b) \leq 0$. Comme par hypothèse $pcost(x : a \to b) \geq 0$, pour chaque $c_S \in \Gamma(x)$, nous avons nécessairement $pcost(c_S, x : a \to b) = 0$. Nous pouvons déduire que pour chaque contrainte $c_S \in \Gamma(x)$, pour chaque instanciation I de S telle que $I[x] = b$ et $c_S(I) = 0$, l'instanciation $I' = I_{x=a}$ est telle que $c_S(I') = 0$. Pour finir, nous pouvons déduire que (x, a) est substituable à (x, b) au voisinage dans $Bool(P)$. □

Proposition 6 *Soient P un WCN VAC-cohérent, $x \in vars(P)$ et $\{a, b\} \subseteq dom(x)$. Si $pcost(x : a \to b) \geq 0$ dans P alors $P \setminus \{(x, b)\}$ est VAC-cohérent.*

Preuve. Supposons (premier cas) pour commencer que (x, b) est AC-cohérent dans $Bool(P)$. D'après le lemme précédent, nous savons que (x, a) est substituable à (x, b) au voisinage dans $Bool(P)$, et donc nous avons $AC(Bool(P)) \neq \bot \Leftrightarrow AC(Bool(P \setminus \{(x, b)\})) \neq \bot$. Puisque P est VAC-cohérent, nécessairement $P \setminus \{(x, b)\}$ est VAC-cohérent. Maintenant (deuxième cas), supposons que (x, b) n'est pas AC-cohérent dans $Bool(P)$. Il est clair que nous avons alors $AC(Bool(P)) = AC(Bool(P \setminus \{(x, b)\}))$, et donc $P \setminus \{(x, b)\}$ est VAC-cohérent (puisque P est VAC-cohérent). □

Corollaire 2 *Soit P un WCN. Si P est VAC-cohérent alors $SNS(P)$ est VAC-cohérent.*

Le corollaire précédent est également valable pour OSAC.

6 Algorithme

Dans cette section, nous présentons un algorithme pour établir AC*+PSNS (qui peut être facilement adapté à EDAC*+PSNS, par exemple). L'idée directrice est de toujours commencer à identifier les valeurs SNS-éliminables à partir d'un WCN qui est AC*-cohérent. Cela nous permet de réduire l'effort de calcul en sortant de boucles avant

Algorithm 1: overcost($c_S, x : a \to b$) : entier

```
ocst ← 1
foreach I ∈ l(S \ {x}) do
    if c_S(I_{x=b}) − c_S(I_{x=a}) < ocst then
        ocst ← c_S(I_{x=b}) − c_S(I_{x=a})
return ocst
```

Algorithm 2: overcost($x, a \to b$) : entier

```
ocst ← c_x(b) − c_x(a)
if ocst < 0 then
    return ocst
ocst ← ocst + overcost(residues[x, a, b], x, a → b)
if ocst < 0 then
    return ocst
foreach c_S ∈ Γ(x) | c_S ≠ residues[x, a, b] do
    d ← overcost(c_S, x, a → b)
    if d < −(c_x(b) − c_x(a)) then
        residues[x, a, b] ← c_S
    ocst ← ocst + d
    if ocst < 0 then
        return ocst
return ocst
```

Algorithm 3: PSNS^r(P : WCN AC*-consistent)

```
Δ ← ∅
foreach x ∈ vars(P) do
    if ∃y ∈ Γ(x) | stamp[y] > substamp then
        foreach (a, b) ∈ dom(x)^2 | b > a do
            if overcost(x, a → b) then
                Δ ← Δ ∪ {(x, b)}
            else if overcost(x, b → a) then
                Δ ← Δ ∪ {(x, a)}
substamp ← time++
foreach (x, a) ∈ Δ do
    remove (x, a) from dom(x)
    Q ← Q ∪ {x}
    stamp[x] ← time++
```

Algorithm 4: AC*-PSNS(P : WCN)

Output: P, rendu AC*-consistent et PSNS-clos

```
time ← 0
substamp ← −1
stamp[x] ← 0, ∀x ∈ vars(P)
Q ← vars(P)
repeat
    PSNS^r(W-AC*(P, Q))
until Q ≠ ∅
```

leur fin et en utilisant des résidus. Pour simplifier, les valeurs SNS-éliminables sont identifiées à l'aide des surcoûts $ocost$ parce que, comme mentionné plus haut, $pcost$ et $ocost$ donnent les mêmes réponses positives ($pcost$ n'étant plus précis que pour les réponses négatives que l'on n'exploite pas ici).

La procédure principale est l'algorithme 4. Comme souvent, nous utilisons un ensemble, noté Q, pour stocker les variables dont le domaine a été récemment réduit. Au début, Q contient toutes les variables (ligne 4). Puis, à la ligne 6, un algorithme classique AC*, désigné ici par W-AC*, est exécuté (par exemple, cela peut être W-AC*2001 [20]), avant de solliciter une fonction appelée PSNS^r. Les appels de W-AC* et de PSNS^r sont entrelacés jusqu'à ce qu'un point fixe soit atteint (i.e., $Q = \emptyset$).

La fonction PSNS^r, algorithme 3, itère sur toutes les variables afin de recueillir les valeurs SNS-éliminable dans un ensemble appelé Δ. Cet ensemble est initialisé à la ligne 1 et mis à jour aux lignes 6 et 8. Imaginons que toutes les valeurs SNS-éliminables (qui peuvent être identifiées avec le calcul de surcoûts) pour une variable x ont été supprimées, et que le domaine de toutes les variables dans le voisinage de x reste identique. Clairement, il n'est alors pas nécessaire de considérer à nouveau x pour rechercher des valeurs SNS-éliminables. C'est l'objet de la ligne 3. Ici, un mécanisme d'horodatage est utilisé. En introduisant un compteur global $time$ et en associant un tampon temporel $stamp[x]$ à chaque variable x ainsi qu'un tampon $substamp$ à la fonction PSNS^r, il est possible de déterminer quelles variables doivent être considérées. La valeur de $stamp[x]$ indique à quel moment une valeur a été récemment éliminée de $dom(x)$ tandis que la valeur de $substamp$ indique à quel moment PSNS^r a été récemment appelé. Les variables $time$, $stamp[x]$ pour chaque variable x et $substamp$ sont initialisées aux ligne 1 à 3 de l'algorithme 4. La valeur de $time$ est incrémentée chaque fois qu'une variable est ajoutée à Q (ligne 13 de l'algorithme 3, et cela *doit* aussi être effectué au sein de W-AC*) et chaque fois que PSNS^r est appelé (ligne 9). Toutes les valeurs SNS-éliminables collectées dans Δ sont supprimées tandis que Q est mis à jour pour pour le prochain appel à W-AC* (lignes 10 à 12).

L'algorithme 2 nous permet de calculer le surcoût $ocost$ de (x, b) par rapport à (x, a). Puisque nous savons que le WCN est AC*-cohérent, nous avons la garantie que le surcoût de (x, b) par rapport à (x, a) dans toute contrainte non-unaire c_S où $x \in S$ est inférieur où égal à 0. Cela signifie que nous ne pourrons jamais compenser une valeur négative avec une valeur positive (une fois que le surcoût unaire a été pris en compte). Un grand avantage de cette observation est la possibilité d'utiliser les arrêts précoces de boucle au cours de tels calculs. Cette opération est effectuée aux lignes 3, 6 et 13. Les résidus sont un autre mécanisme introduit pour augmenter la performance de l'algorithme. Pour chaque variable x, et pour tout couple (a, b) de valeurs de $dom(x)$, nous stockons dans $residues[x, a, b]$ la contrainte c_S qui garantit que (x, b) n'est pas SNS-éliminable par (x, a), si elle existe. La contrainte résiduelle est prioritaire (lignes 4 à 6) ; de cette façon, si elle permet de compenser le surcoût initial unaire, elle nous évite tout travail supplémentaire. Elle est mise à jour aux lignes 9-10. Notons que nous pouvons initialiser le tableau $residues$ avec n'importe quelle contrainte arbitraire et que l'algorithme 1 retourne nécessairement une valeur inférieure ou égale à 0 (ce qui explique l'initialisation de $ocst$ à 1 à la ligne 1).

Nous discutons maintenant de la complexité de PSNS^r tout en faisant l'hypothèse (pour simplifier) que le WCN est binaire. La complexité en espace est $O(nd^2)$ en raison de l'utilisation de la structure $residues$. La complexité en temps de l'algorithme 1 est $O(d)$, et la complexité en temps de l'algorithme 2 est $O(1)$ dans le meilleur des cas (si il est arrêté à la ligne 3) et $O(qd)$ dans le pire des cas, où $q = |\Gamma(x)|$. Sans la ligne 3, la complexité en temps de l'algorithme 3 est $O(nd^2)$ dans le meilleur des cas et $O(nd^3e)$ dans le pire des cas (à noter que $\sum_{x \in vars(P)} |\Gamma(x)|$ est $O(e)$). Bien sûr, l'algorithme 3 peut être appelé à plusieurs reprises à la ligne 6 de l'algorithme 4, ainsi obtient-on une complexité dans le pire des cas en $O(n^2d^4e)$. Cependant, nous avons observé dans nos expérimentations que le nombre d'appels successifs à PSNS^r était très limité en pratique (comme on l'imaginait). En outre, imaginons maintenant que nous appelions AC*-PSNS après l'assignation d'une valeur à une variable (i.e. au cours de la recherche) x. Dans le meilleur des cas (du point de vue de la complexité temporelle), aucune suppression n'est effectué par $W\text{-}AC*$, et on considère donc uniquement le voisinage de x (à la ligne 3 de l'algorithme 3), ce qui donne une complexité en temps en $O(qd^2)$. Ce dernier résultat nous permet d'envisager relativement sereinement l'expérimentation du maintien de AC*-PSNS pendant la recherche.

7 Résultats expérimentaux

Pour démontrer l'intérêt pratique de la suppression des valeurs SNS-éliminables, nous avons mené une expérimentation en utilisant les séries d'instances WCSP disponible à l'adresse `http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/BenchmarkS` et un cluster de Xeon 3.0GHz avec 2 Go de RAM sous Linux. Notre but est d'observer l'efficacité relative de la résolution d'instances WCSP lorsqu'on *maintient* AC*, AC*+PSNS, FDAC, FDAC+PSNS, EDAC, et EDAC+PSNS. Pour l'ordre de sélection des variables au cours de la recherche, nous utilisons l'heuristique statique simple *max degree* qui est indépendante de l'efficacité des algorithmes de filtrage utilisés, comme dans [11] où certaines expériences ont été réalisées avec une forme partielle de SNS appliqué lors d'une étape de pré-traitement.

Series		AC*	AC* +PSNS	FDAC	FDAC +PSNS	EDAC	EDAC +PSNS
celar	#solved	5	4	6	6	6	**7**
#inst=7	cpu	337	231	316	341	344	**461**
	avg-sub	0	3	0	6	0	4
driver	#solved	18	18	19	19	19	**19**
#inst=19	cpu	103	52	39.3	19.7	68.5	**56.8**
	avg-sub	0	1	0	1	0	1
geom	#solved	5	5	5	**5**	5	5
#inst=5	cpu	37.4	12.4	18.8	**11.0**	21.0	12.0
	avg-sub	0	0	0	1	0	0
mprime	#solved	4	8	4	**8**	5	8
#inst=8	cpu	9.82	17.4	11.6	**12.0**	206	21.0
	avg-sub	0	1	0	1	0	1
myciel	#solved	3	3	3	**3**	3	3
#inst=3	cpu	122	77.0	72.9	**34.3**	80.4	37.8
	avg-sub	0	2	0	3	0	3
scens+graphs	#solved	1	3	**8**	7	6	5
#inst=9	cpu	20.4	113	**242**	186	266	19.8
	avg-sub	0	8	0	8	0	8
spot5	#solved	0	0	3	3	3	**3**
#inst=3	cpu			20.5	12.6	20.1	**11.4**
	avg-sub	0	0	0	0	0	0
warehouse	#solved	12	12	18	24	28	**29**
#inst=34	cpu	286	46.2	166	139	53.2	**79.8**
	avg-sub	0	4	0	7	0	27
	#solved	48	53	66	75	75	**79**

TABLE 1 – Résultats obtenus pour différentes séries (une échéance de 1,200 secondes par instance).

Le tableau 2 montre les moyennes des résultats obtenus sur les différentes séries. Pour chaque série, le nombre d'instances considérées (#inst) est donné au-dessous du nom de la série. Nous avons écarté les instances qui n'ont pas été résolues par au moins un des algorithmes, en moins de 1,200 secondes. Ici, une instance résolue signifie qu'une solution optimale a été trouvée et prouvée être optimale. Le nombre moyen (avg-sub) de valeurs SNS-éliminables supprimées lors de la recherche (à chaque étape) est également indiqué (avec des valeurs arrondies à l'entier le plus proche). Sur les instances RLFAP (CELAR, scens, graphs), l'intérêt d'utiliser PSNS est plutôt chaotique, mais sur les instances (driver, mprime), coloring (myciel, geom), spot et warehouse, on peut constater qu'il y a un avantage clair à l'intégration de PSNS. Dans l'ensemble, le maintien de PSNS est rentable car il offre en général un avantage à la fois en termes d'instance résolues (voir dernière ligne du tableau) et en temps CPU. Le tableau 2 présente les résultats obtenus sur certaines instances représentatives. Il est intéressant de noter que sur les instances warehouse (ici, *cap101*, *cap111* et *capmo1*), l'application de PSNS

Instances		AC*	AC* +PSNS	FDAC	FDAC +PSNS	EDAC	EDAC +PSNS
cap101	cpu	232	42.1	1.6	1.62	1.48	**1.12**
	#nodes	242K	242K	835	835	75	75
	avg-sub	0	1	0	2	0	15
cap111	cpu	>1,200	>1,200	633	162	3.03	**2.74**
	#nodes	–	–	72,924	72,924	439	199
	avg-sub	0	2	0	3	0	12
capmo1	cpu	>1,200	>1,200	>1,200	>1,200	>1,200	**984**
	#nodes	–	–	–	–	–	–
	avg-sub	0	15	0	23	0	64
driverlog02ac	cpu	253	49.5	10.6	**7.99**	19.3	13.5
	#nodes	4,729K	701K	19,454	8,412	19,444	8,402
	avg-sub	0	0	0	0	0	0
driverlogs06	cpu	>1,200	>1,200	187	**61.0**	218	80.1
	#nodes	–	–	2,049K	609K	2,049K	609K
	avg-sub	0	0	0	0	0	0
mprime04ac	cpu	>1,200	30.9	>1,200	**15.7**	>1,200	43.7
	#nodes	–	189K	–	22,373	–	20,381
	avg-sub	0	0	0	1	0	1
myciel5g-3	cpu	38.1	19.6	**3.87**	4.18	4.36	4.57
	#nodes	518K	168K	10,046	9,922	6,159	6,128
	avg-sub	0	2	0	3	0	2
celar7-sub1	cpu	925	820	147	145	135	**86.4**
	#nodes	9,078K	1,443K	732K	91,552	796K	70,896
	avg-sub	0	6	0	9	0	6
graph07	cpu	>1,200	261	**3.11**	3.58	3.86	4.3
	#nodes	6,156K	145K	1,112	647	1,796	1,514
	avg-sub	0	23	0	9	0	4
scen06-24	cpu	>1,200	>1,200	**1,061**	>1,200	>1,200	>1,200
	#nodes	–	–	–	375K	–	–
	avg-sub	0	2	0	6	0	7
spot5-29	cpu	>1,200	>1,200	30.1	**18.0**	47.2	22.5
	#nodes	–	–	343K	174K	352K	185K
	avg-sub	0	0	0	0	0	0

TABLE 2 – Résultats illustratifs obtenus sur certaines instances.

n'entraîne pas une réduction de la taille de l'arbre de recherche (voir les valeurs de #nodes). Cependant, PSNS permet de réduire la taille des domaines, ce qui rend la propagation des contraintes souples plus rapide. Sur *celar7-sub1*, notons que (maintenir) EDAC+PSNS est seulement environ 50% plus rapide que (maintenir) EDAC alors que le nombre de nœuds a été divisé par 10. Cela signifie que sur de telles instances, PSNS est assez coûteux, ce qui laisse sans doute la place à des optimisations supplémentaires.

8 Conclusion

Dans cet article, nous avons étudié la propriété de substituabilité souple au voisinage pour les réseaux de

contraintes pondérées (WCNs) et analysé les conditions permettant l'identification de valeurs substituables par un algorithme de complexité raisonnable (i.e., polynomial). Nous avons prouvé que, même dans les cas simples, lorsque $k \neq \infty$, le problème de décider si une valeur est souple-substituable à une autre au voisinage est coNP-difficile. Nous avons également étudié les relations entre substituabilité souple au voisinage et cohérence d'arc souple. Enfin, nous avons proposé un algorithme qui exploite des arrêts précoces de boucles, des résidus et un mécanisme d'horodatage, et montré expérimentalement qu'il peut être efficace au cours de la recherche.

Remerciements

Ce travail bénéficie du soutien du CNRS et d'OSEO dans le cadre du projet ISI Pajero.

Références

[1] A. Bellicha, C. Capelle, M. Habib, T. Kokény, and M.C. Vilarem. CSP techniques using partial orders on domain values. In *Proceedings of ECAI'94 workshop on constraint satisfaction issues raised by practical applications*, 1994.

[2] B.W. Benson and E.C. Freuder. Interchangeability preprocessing can improve forward checking search. In *Proceedings of ECAI'92*, pages 28–30, 1992.

[3] S. Bistarelli, B. Faltings, and N. Neagu. Interchangeability in soft CSPs. In *Proceedings of CSCLP'02*, pages 31–46, 2002.

[4] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs : Frameworks, properties, and comparison. *Constraints*, 4(3) :199–240, 1999.

[5] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Support inference for generic filtering. In *Proceedings of CP'04*, pages 721–725, 2004.

[6] B.Y. Choueiry, B. Faltings, and R. Weigel. Abstraction by interchangeability in resource allocation. In *Proceedings of IJCAI'95*, pages 1694–1710, 1995.

[7] M. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3) :311–342, 2003.

[8] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, and M. Zytnicki. Virtual arc consistency for weighted CSP. In *Proceedings of AAAI'08*, pages 253–258, 2008.

[9] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8) :449–478, 2010.

[10] M.C. Cooper. Fundamental properties of neighbourhood substitution in constraint satisfaction problems. *Artificial Intelligence*, 90 :1–24, 1997.

[11] S. de Givry. Singleton consistency and dominance for weighted CSP. In *Proceedings of CP'04 Workshop on Preferences and Soft Constraints*, 2004.

[12] S. de Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency : Getting closer to full arc consistency in weighted CSPs. In *Proceedings of IJCAI'05*, pages 84–89, 2005.

[13] E.C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI'91*, pages 227–233, 1991.

[14] E.C. Freuder and D. Sabin. Interchangeability supports abstraction and reformulation for multi-dimensional constraint satisfaction. In *Proceedings of AAAI'97*, pages 191–196, 1997.

[15] A. Haselbock. Exploiting interchangeabilities in constraint satisfaction problems. In *Proceedings of IJCAI'93*, pages 282–287, 1993.

[16] S. Karakashian, R. Woodward, B. Choueiry, S. Prestwhich, and E. Freuder. A partial taxonomy of substitutability and interchangeability. Technical Report arXiv :1010.4609, CoRR, 2010.

[17] A.M. Koster. *Frequency assignment : Models and Algorithms*. PhD thesis, University of Maastricht, The Netherlands, 1999.

[18] A. Lal, B.Y. Choueiry, and E.C. Freuder. Neighborhood interchangeability and dynamic bundling for non-binary finite CSPs. In *Proceedings of AAAI'05*, pages 397–404, 2005.

[19] J. Larrosa. Node and arc consistency in weighted CSP. In *Proceedings of AAAI'02*, pages 48–53, 2002.

[20] J. Larrosa and T. Schiex. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, 159(1-2) :1–26, 2004.

[21] S. Martello and P. Toth. *Knapsack Problems : Algorithms and Computer Implementations*. Wiley, 1990.

[22] A. Petcu and B. Faltings. Applying interchangeability techniques to the distributed breakout algorithm. In *Proceedings of CP'03*, pages 925–929, 2003.

[23] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems : Hard and easy problems. In *Proceedings of IJCAI'95*, pages 631–639, 1995.

Analyse Syntaxique par Contraintes pour les Grammaires de Propriétés à Traits

Denys Duchier **Thi-Bich-Hanh Dao** **Yannick Parmentier**

Laboratoire d'Informatique Fondamentale d'Orléans – Université d'Orléans
Bâtiment 3IA, Rue Léonard de Vinci – 45067 Orléans Cedex 2
prenom.nom@univ-orleans.fr

Résumé

Les Grammaires de Propriétés (GP) constituent un formalisme à base de contraintes capable de décrire la syntaxe, à la fois d'énoncés bien-formés et d'énoncés agrammaticaux. Duchier *et al.* (2010) proposent une sémantique formelle des GP en théorie des modèles et modélisent l'analyse syntaxique en GP sous la forme d'un Problème de Satisfaction de Contraintes (CSP). Dans cet article, nous présentons une extension de ces travaux afin de manipuler de nouveaux types de propriétés, qui permettent d'exprimer des relations entre constituants syntaxiques sous forme de contraintes sur structures de traits (*e.g.* la propriété d'*accord*). Nous décrivons ensuite une extension du modèle CSP pour traiter ces nouveaux types de propriétés lors de l'analyse syntaxique.

1 Introduction

Une des tâches spécifiques du Traitement Automatique des Langues est l'analyse syntaxique. Cette tâche a pour but de construire, à partir d'une description formelle de la syntaxe de la langue naturelle (*i.e.*, une grammaire), une structure exprimant les relations entre les divers constituants d'un énoncé. Cette structure prend généralement une forme arborescente, on parle alors d'arbre syntaxique.

De nombreux formalismes grammaticaux ont été proposés pour décrire la syntaxe de la langue naturelle, généralement en se basant sur des systèmes de réécriture. Un problème majeur de ces formalismes est leur manque de robustesse. En effet, ils ne permettent pas d'analyser des énoncés qui sont mal-formés, même lorsqu'il s'agit d'erreurs mineures. Les grammaires formelles de type *syntaxe fondée sur la théorie des modèles*, qui se concentrent sur une validation de modèles en terme de contraintes satisfaites, par contre, sont naturellement adaptées au traitement de *quasi-expressions* (*i.e.*, d'énoncés agrammaticaux produits par l'Homme).

Blache [1] a proposé le formalisme des Grammaires de Propriétés (GP), qui est un formalisme à base de contraintes, pour décrire à la fois des énoncés grammaticaux et agrammaticaux. Dans [2], Duchier *et al.* proposent une sémantique formelle des GP en théorie des modèles et modélisent l'analyse syntaxique en GP sous la forme d'un Problème de Satisfaction de Contraintes (CSP). Cependant, ces travaux ne permettent d'exprimer que des contraintes entre constituants syntaxiques relativement grossières (*e.g.*, exclusion mutuelle entre constituants de certains types, *etc.*). Ces contraintes ne permettent pas d'exprimer des restrictions plus fines, c'est-à-dire des contraintes sur des structures de traits, telles que l'accord entre constituants par exemple. Dans notre travail, nous étendons la sémantique des GP pour pouvoir exprimer ces contraintes sur des structures de traits dans un cadre formel en théorie des modèles. En plus de la définition formelle de nouveaux types de contraintes, notre travail aborde l'extension de la modélisation de l'analyse syntaxique sous forme de CSP de Duchier *et al.* afin de traiter ces nouvelles contraintes.

2 Grammaires de Propriétés (GP)

Les grammaires de propriétés [1] constituent un formalisme permettant de décrire la langue naturelle en terme de contraintes locales, appelées *propriétés*. Ces propriétés peuvent être violées indépendamment les unes les autres, ce qui rend possible la description d'énoncés agrammaticaux et également une notion de grammaticalité (ratio entre propriétés violées et

propriétés vérifiées). En première approximation, nous pouvons considérer des propriétés de la forme $A : \psi$, spécifiant que, pour chaque nœud de catégorie A, la contrainte ψ s'applique sur les nœuds fils de A (notés B, C ci-dessous). La contrainte ψ a l'une des formes suivantes :

Obligation	$A : \triangle B$	au moins un B
Unicité	$A : B!$	au plus un B
Linéarité	$A : B \prec C$	B précède C
Exigence	$A : B \Rightarrow C$	si $\exists$ B, alors $\exists$ C
Exclusion	$A : B \not\Rightarrow C$	pas B et C
Constituence	$A : S?$	tout enfant $\in S$
Accord	$A : B \rightsquigarrow C$	accord entre B et C

En pratique, la description des langues naturelles n'assigne pas une catégorie atomique à un constituant syntaxique, mais plutôt, une structure de traits (appelée aussi matrice attributs-valeurs), contenant des informations diverses telles que le genre, le temps, *etc.* Dans un arbre syntaxique, chaque nœud est donc étiqueté non pas par une unique catégorie, mais par un ensemble de *traits* (incluant un trait *catégorie*). Pour avoir une description fine sous forme de propriétés, nous devons donc étendre le formalisme pour imposer des contraintes entre traits. Commençons par décrire formellement ces structures de traits.

Soit $\mathcal{F}$ un ensemble fini de traits $\{f_1, \ldots, f_n\}$, où chaque f_i prend sa valeur dans un semi-treillis borné D_i. On note $\top_i$ pour le plus grand élément de D_i. On suppose que parmi les traits f_i, il existe un trait `cat` désignant la catégorie syntaxique. On considère des matrices attributs-valeurs (AVM) de type $\mathcal{M} = [f_1{:}D_1, \ldots, f_n{:}D_n]$. On note $M|_{f_i}$ pour la valeur du trait f_i dans M. Les AVMs forment aussi un semi-treillis avec l'ordre $M_1 \sqsubseteq M_2$ ssi $\forall i$, $M_1|_{f_i} \sqsubseteq M_2|_{f_i}$. On omettra f_i si sa valeur est $\top_i$. Une expression-AVM est une AVM dans laquelle des traits peuvent être associés à des variables partagées (contraintes de co-référence). Dans ce contexte, si S est une expression-AVM (*i.e.*, une AVM contenant des traits dont les valeurs sont liées par l'utilisation de variables partagées, notées dans ce qui suit $X, Y, \ldots$), alors S^v est une AVM obtenue en remplaçant, dans S, chaque occurrence de $f_i{:}X$ par $f_i{:}\top_i$. Cette notion de valeur d'AVM S^v nous sera utile dans les cas où l'on souhaite ignorer les contraintes de co-référence (voir Section 3). De plus, si S_0, S_1, S_2 sont des expressions-AVM, alors $E(S_0, S_1, S_2)$ désigne l'ensemble des équations de co-référence $(S_i|_f{:}X, S_j|_g{:}X)$, que nous notons $(i, f) \doteq (j, g)$, pour tout attribut f dans S_i et g dans S_j partageant une variable X, avec $0 \leq i \leq 2$, $0 \leq j \leq 2$, et $f, g \in \mathcal{F}$. Ce concept d'ensemble d'équations $E(S_0, S_1, S_2)$ nous sera utile dans les cas où l'on souhaite imposer des contraintes de co-référence entre traits appartenant à des triplets d'AVMs S_0, S_1, S_2 (voir Section 3).

À partir de cette définition des structures de traits, nous pouvons étendre les propriétés de GP définies ci-dessus respectivement comme suit : $S_0 : \triangle S_1$, $S_0 : S_1!$, $S_0 : S_1 \prec S_2$, $S_0 : S_1 \Rightarrow S_2$, $S_0 : S_1 \not\Rightarrow S_2$, $S_0 : s_1?$, $S_0 : S_1 \rightsquigarrow S_2$, où S_0, S_1, S_2 sont des expressions-AVM (et s_1 un ensemble d'expressions-AVM). Pour illustrer ces propriétés à traits, prenons l'exemple de la propriété d'accord. Celle-ci peut être utilisée pour modéliser, au sein d'un groupe verbal, l'accord en genre, nombre et personne entre le participe passé du verbe et un complément d'objet direct lorsque celui-ci est réalisé avant le verbe (*je l'ai aimée*) :

$$\texttt{GV} : \texttt{V}\begin{bmatrix}\text{mode} & \text{part-passé}\\ \text{genre} & \text{X}\\ \text{nombre} & \text{Y}\\ \text{pers} & \text{Z}\end{bmatrix} \rightsquigarrow \texttt{Pro}\begin{bmatrix}\text{cas} & \text{COD}\\ \text{genre} & \text{X}\\ \text{nombre} & \text{Y}\\ \text{pers} & \text{Z}\end{bmatrix}$$

Dans cet exemple, le trait `cat` (de valeur `GV`, `V` et `Pro`) est extrait des matrices afin de respecter les conventions de notation en Grammaires de Propriétés.

Soit $\mathcal{W}$ un ensemble de *mots*. Un lexique est un sous-ensemble de $\mathcal{W} \times \mathcal{M}$ (un lexique associe donc à chaque mot une ou plusieurs AVMs). Une *grammaire de propriétés* G est un couple (P_G, L_G), où P_G est un ensemble de propriétés et L_G un lexique.

3 Sémantique formelle des GP

Nous allons maintenant étendre la spécification formelle en théorie des modèles des GP proposées par Duchier *et al.* [3] afin de tenir compte des propriétés à traits introduites précédemment.

Classe de modèles. Comme dans [3], la sémantique des grammaires de propriétés est donnée par une interprétation sur la structure des arbres syntaxiques τ. Cette structure est définie comme suit. Nous notons $\mathbb{N}_0$ l'ensemble $\mathbb{N}\backslash\{0\}$. Un *domaine d'arbre* D est un sous-ensemble fini de $\mathbb{N}_0^*$ (*i.e.*, l'ensemble des mots construits sur le vocabulaire $\mathbb{N}_0$) clos pour les préfixes et les frères-gauches. En d'autres termes, $\forall \pi, \pi' \in \mathbb{N}_0^*, \forall i, j \in \mathbb{N}_0$:

$$\begin{array}{lcl} \pi\pi' \in D & \Rightarrow & \pi \in D \\ i < j \wedge \pi j \in D & \Rightarrow & \pi i \in D \end{array}$$

Un arbre syntaxique $\tau = (D_\tau, L_\tau, R_\tau)$ consiste en un domaine d'arbre D_τ, une fonction d'étiquetage $L_\tau : D_\tau \to \lfloor \mathcal{M} \rfloor$ associant à chaque nœud une AVM *minimale* (pour $\sqsubseteq$) et une fonction $R_\tau : D_\tau \to \mathcal{W}^*$ associant à chaque nœud sa réalisation surfacique. Pour des raisons pratiques, nous définissons aussi la fonction

d'arité $A_\tau : D_\tau \to \mathbb{N}$ comme suit, $\forall \pi \in D_\tau$:

$$A_\tau(\pi) = \max\{0\} \cup \{i \in \mathbb{N}_0 \mid \pi i \in D_\tau\}$$

Instances de propriété. Comme dans [3], nous définissons des instances de propriété p sur un arbre τ (notées $\mathcal{I}_\tau[\![p]\!]$), correspondant à des paires $X \,@\, Y$, où X est une propriété telle que présentée en Section 2, et Y un n-uplet de nœuds (*i.e.*, de chemins) sur lesquelles s'applique la propriété (voir Fig. 1).

$$\begin{aligned}
\mathcal{I}_\tau[\![G]\!] &= \cup\{\mathcal{I}_\tau[\![p]\!] \mid \forall p \in P_G\} \\
\mathcal{I}_\tau[\![S_0 : S_1 \prec S_2]\!] &= \{(S_0 : S_1 \prec S_2)@\langle \pi, \pi i, \pi j\rangle \mid \forall \pi, \pi i, \pi j \in D_\tau, i{\neq}j\} \\
\mathcal{I}_\tau[\![S_0 : \triangle S_1]\!] &= \{(S_0 : \triangle S_1)@\langle \pi\rangle \mid \forall \pi \in D_\tau\} \\
\mathcal{I}_\tau[\![S_0 : S_1!]\!] &= \{(S_0 : S_1!)@\langle \pi, \pi i, \pi j\rangle \mid \forall \pi, \pi i, \pi j \in D_\tau, i{\neq}j\} \\
\mathcal{I}_\tau[\![S_0 : S_1 \Rightarrow S_2]\!] &= \{(S_0 : S_1 \Rightarrow S_2)@\langle \pi, \pi i\rangle \mid \forall \pi, \pi i \in D_\tau\} \\
\mathcal{I}_\tau[\![S_0 : S_1 \not\Rightarrow S_2]\!] &= \{(S_0 : S_1 \not\Rightarrow S_2)@\langle \pi, \pi i, \pi j\rangle \mid \forall \pi, \pi i, \pi j \in D_\tau, i{\neq}j\} \\
\mathcal{I}_\tau[\![S_0 : s_1?]\!] &= \{(S_0 : s_1?)@\langle \pi, \pi i\rangle \mid \forall \pi, \pi i \in D_\tau\} \\
\mathcal{I}_\tau[\![S_0 : S_1 \rightsquigarrow S_2]\!] &= \{(S_0 : S_1 \rightsquigarrow S_2)@\langle \pi, \pi i, \pi j\rangle \mid \forall \pi, \pi i, \pi j \in D_\tau, i{\neq}j\}
\end{aligned}$$

FIGURE 1 – Instances de propriété d'une grammaire G sur un arbre τ

Par exemple, sur la Fig. 1, la seconde ligne indique qu'une propriété de linéarité s'instancie sur un triplet de nœuds $\langle \pi, \pi i, \pi j\rangle$ composé d'un nœud père et de deux nœuds fils.

Pertinence. Nous avons ainsi une instance pour chaque propriété de P_G et chaque n-uplet de nœuds de τ. Nous devons en outre distinguer les instances des propriétés pertinentes de celles qui ne le sont pas, pour cela nous introduisons un prédicat P_τ (voir Fig. 2). Cette définition de la pertinence étend celle de [3] en y intégrant des comparaisons entre AVMs.

$$\begin{aligned}
&P_\tau((S_0 : S_1 \prec S_2)@\langle \pi, \pi i, \pi j\rangle) \equiv \\
&\qquad (L_\tau(\pi) \sqsubseteq S_0^v) \ \wedge\ (L_\tau(\pi i) \sqsubseteq S_1^v) \ \wedge\ (L_\tau(\pi j) \sqsubseteq S_2^v) \\
&P_\tau((S_0 : \triangle S_1)@\langle \pi\rangle) \equiv \\
&\qquad L_\tau(\pi) \sqsubseteq S_0^v \\
&P_\tau((S_0 : S_1!)@\langle \pi, \pi i, \pi j\rangle) \equiv \\
&\qquad (L_\tau(\pi) \sqsubseteq S_0^v) \ \wedge\ (L_\tau(\pi i) \sqsubseteq S_1^v) \ \wedge\ (L_\tau(\pi j) \sqsubseteq S_1^v) \\
&P_\tau((S_0 : S_1 \Rightarrow S_2)@\langle \pi, \pi i\rangle) \equiv \\
&\qquad (L_\tau(\pi) \sqsubseteq S_0^v) \ \wedge\ (L_\tau(\pi i) \sqsubseteq S_1^v) \\
&P_\tau((S_0 : S_1 \not\Rightarrow S_2)@\langle \pi, \pi i, \pi j\rangle) \equiv \\
&\qquad (L_\tau(\pi) \sqsubseteq S_0^v) \ \wedge\ ((L_\tau(\pi i) \sqsubseteq S_1^v) \ \vee\ (L_\tau(\pi j) \sqsubseteq S_2^v)) \\
&P_\tau((S_0 : s_1?)@\langle \pi, \pi i\rangle) \equiv \\
&\qquad L_\tau(\pi) \sqsubseteq S_0^v \\
&P_\tau((S_0 : S_1 \rightsquigarrow S_2)@\langle \pi, \pi i, \pi j\rangle) \equiv \\
&\qquad (L_\tau(\pi) \sqsubseteq S_0^v) \ \wedge\ (L_\tau(\pi i) \sqsubseteq S_1^v) \ \wedge\ (L_\tau(\pi j) \sqsubseteq S_2^v)
\end{aligned}$$

FIGURE 2 – Pertinence d'instances sur un arbre τ

Ainsi, sur la Fig. 2, nous remarquons que pour évaluer la pertinence d'une propriété, nous ne tenons pas compte des éventuelles co-références entre traits. Nous vérifions uniquement que les AVMs étiquetant les nœuds concernés ont des traits dont les valeurs sont compatibles avec la propriété (pour la relation $\sqsubseteq$).

Satisfaction. Lorsqu'une instance est pertinente, elle peut également être satisfaite. Pour définir cette notion de satisfaction, nous étendons le prédicat S_τ de [3] comme illustré en Fig. 3. Par exemple, la dernière ligne de la Fig. 3 définit la satisfaction pour la propriété d'accord (la seule manipulant des contraintes de co-référence entre traits). Cettte satisfaction dépend de la satisfaction d'équations de co-référence. Soient M_0, M_1, M_2 des AVMs, celles-ci satisfont les équations de co-référence de S_0, S_1, S_2 (noté $M_0, M_1, M_2 \models E(S_0, S_1, S_2)$), ssi $M_i|_f{=}M_j|_g$ pour tout $(i, f)\doteq(j, g)$ dans $E(S_0, S_1, S_2)$. Pour la propriété de linéarité (première ligne de la Fig. 3) par contre, la satisfaction n'est conditionnée que par l'ordre entre les nœuds fils du noeud π, sur lesquels s'instancie la propriété.

$$\begin{aligned}
&S_\tau((S_0 : S_1 \prec S_2)@\langle \pi, \pi i, \pi j\rangle) \equiv \\
&\qquad i < j \\
&S_\tau((S_0 : \triangle S_1)@\langle \pi\rangle) \equiv \\
&\qquad \vee\{(L_\tau(\pi i) \sqsubseteq S_1^v) \mid 1 \le i \le A_\tau(\pi)\} \\
&S_\tau((S_0 : S_1!)@\langle \pi, \pi i, \pi j\rangle) \equiv \\
&\qquad i = j \\
&S_\tau((S_0 : S_1 \Rightarrow S_2)@\langle \pi, \pi i\rangle) \equiv \\
&\qquad \vee\{(L_\tau(\pi j) \sqsubseteq S_2^v) \mid 1 \le j \le A_\tau(\pi)\} \\
&S_\tau((S_0 : S_1 \not\Rightarrow S_2)@\langle \pi, \pi i, \pi j\rangle) \equiv \\
&\qquad (L_\tau(\pi i) \not\sqsubseteq S_1^v) \ \vee\ (L_\tau(\pi j) \not\sqsubseteq S_2^v) \\
&S_\tau((S_0 : s_1?)@\langle \pi, \pi i\rangle) \equiv \\
&\qquad L_\tau(\pi i) \sqsubseteq x \quad \text{for some } x \text{ in } s_1 \\
&S_\tau((S_0 : S_1 \rightsquigarrow S_2)@\langle \pi, \pi i, \pi j\rangle) \equiv \\
&\qquad L_\tau(\pi), L_\tau(\pi i), L_\tau(\pi j) \models E(S_0, S_1, S_2)
\end{aligned}$$

FIGURE 3 – Satisfaction d'instances sur un arbre τ

Rappelons que l'intérêt de définir ces relations de pertinence et satisfaction d'instance, est, comme le précise [3], de pouvoir ensuite calculer un score reflétant le degré de grammaticalité d'un arbre syntaxique. Ainsi, un arbre syntaxique τ est un modèle de G s'il maximise le ratio $I^+_{G,\tau}/I^0_{G,\tau}$, où $I^0_{G,\tau}$ représente le nombre de contraintes pertinentes, et $I^+_{G,\tau}$ celui des contraintes pertinentes et satisfaites.

4 Analyse syntaxique des GP avec traits

À présent, nous montrons comment convertir cette sémantique de GP avec traits sous forme d'un analyseur par CSP, en étendant l'approche de Duchier *et al.* [2], qui utilise le concept d'arbres rectangulaires. Rappelons simplement ici, que ces arbres rectangulaires sont des arbres dont les nœuds sont placés de manière univoque dans une grille de taille fixée (suppression des symétries). Cette grille permet de plus de restreindre le nombre de nœuds composant un modèle bien que ce nombre soit *a priori* inconnu.

Pour un énoncé de m mots, nous savons que chaque modèle a m nœuds feuilles. Par contre, nous ne connaissons pas la hauteur de ces modèles, nous décidons de faire de cette hauteur n un paramètre du

CSP. Ainsi, chaque modèle est représenté par une matrice $\mathcal{W}$ telle que w_{ij} (avec $1 \leq i \leq n$, et $1 \leq j \leq m$) réfère au nœud situé à la position (i, j) sur la grille (la coordonnée (1,1) étant située dans le coin en bas à gauche). La Fig. 4 illustre ceci au moyen de l'arbre syntaxique de la phrase "Pierre mange la pomme".

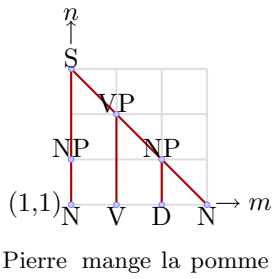


FIGURE 4 – Arbre syntaxique placé sur une grille

Notons que sur une telle grille, nous distinguons deux types de nœuds, ceux qui sont utilisés par un modèle (dits nœuds actifs), et ceux qui ne le sont pas (nœuds inactifs). Soit V l'ensemble des nœuds, V^+ est l'ensemble des nœuds actifs et V^- celui des nœuds inactifs. Un nœud est soit actif soit (strictement) inactif : $V = V^+ \uplus V^-$ ($\uplus$ étant l'union disjointe). La liste des contraintes permettant de produire de telles arbres rectangulaires est donnée dans [2].

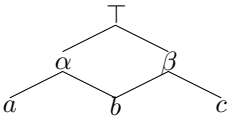
Représenter des contraintes sur semi-treillis. Comme cela a été présenté en Section 3, les contraintes sur structures de traits sont définies formellement sous forme de contraintes sur des AVMs dont les traits prennent leur valeur dans des semi-treillis. Voyons comment encoder ces contraintes sous forme de contraintes ensemblistes (notons que cet encodage relativement simple pourrait être étendu en utilisant des techniques telles que celles proposées par *Fall* [4]).

Dans les définitions de la *pertinence* et de la *pertinence et satisfaction*, nous devons représenter la relation $L_\tau(w) \sqsubseteq S^v$, où S est une expression-AVM et S^v est obtenue en remplaçant, dans S, chaque occurrence de $f{:}X$, où X est une variable, par $f{:}\top$. S^v est donc une AVM $[f_1{:}v_1, \ldots, f_n{:}v_n]$ sans variable, telle que chaque $v_i = S^v|_{f_i}$ ait sa valeur dans D_i.

Puisque $L_\tau(w)$ doit être instanciée par une AVM *minimale*, si $L_\tau(w) \sqsubseteq S^v$ alors chaque $L_\tau(w)|_{f_i}$ doit être associé avec une constante minimale $c_i \in D_i$ (*i.e.* un élément de D_i minimal pour l'ordre partiel $\sqsubseteq$), et telle que $c_i \sqsubseteq v_i$. Nous notons $\lfloor v_i \rfloor$ l'ensemble des valeurs minimales plus petites ($\sqsubseteq$) que v_i dans D_i. La relation $L_\tau(w) \sqsubseteq S^v$ peut alors être représentée par :

$$\bigwedge_{i=1}^{n} L_\tau(w)|_{f_i} \in \lfloor S^v|_{f_i} \rfloor \qquad (1)$$

Considérons par exemple une AVM $S^v = [f{:}\alpha]$, et le semi-treillis suivant.



Dans ce contexte, $L_\tau(w)$ satisfait la relation $L_\tau(w) \sqsubseteq S^v$ ssi $L_\tau(w)|_f$ appartient à $\lfloor \alpha \rfloor = \{a, b\}$.

La co-référence se représente comme suit. Soient S_0, S_1, S_2 des expressions-AVM. Rappelons que l'ensemble des équations de co-référence $E(S_0, S_1, S_2)$ est l'ensemble des équations $(i, f_u) \doteq (j, f_v)$ pour tout $f_u{:}X$ dans S_i et $f_v{:}X$ dans S_j. La relation $L_\tau(w_0), L_\tau(w_1), L_\tau(w_2) \models E(S_0, S_1, S_2)$ est représentée par :

$$\bigwedge_{(i,f_u)\doteq(j,f_v)\in E(S_0,S_1,S_2)} L_\tau(w_i)|_{f_u} = L_\tau(w_j)|_{f_v} \qquad (2)$$

Si nous retournons à notre problème de calcul de modèles d'arbres syntaxiques, nous devons assigner à chaque nœud actif w une étiquette $L_\tau(w)$, qui est une AVM minimale (pour $\sqsubseteq$). On pourrait modéliser $L_\tau(w)|_{f_i}$ au moyen *d'une variable de domaine fini* sur les valeurs minimales de D_i, mais alors que faire des nœuds inactifs ? Nous décidons plutôt de modéliser $L_\tau(w)|_{f_i}$ par une *variable ensembliste* de cardinalité au plus 1 :

$$L_\tau(w)|_{f_i} \subseteq \lfloor \top_i \rfloor \qquad |L_\tau(w)|_{f_i}| \leq 1$$

$$w \in V^+ \quad \Leftrightarrow \quad |L_\tau(w)|_{f_i}| = 1 \qquad \forall i \in [1, n]$$

et la relation de subsomption $L_\tau(w) \sqsubseteq S^v$ de (1) est alors représentée par :

$$\bigwedge_{i=1}^{n} L_\tau(w)|_{f_i} \subseteq \lfloor S^v|_{f_i} \rfloor \qquad (3)$$

Enfin, les nœuds feuilles de notre modèle doivent pouvoir être reliés aux mots du lexique L_G constitué de paires (mot, AVM) :

$$\bigwedge_{j=1}^{m} \exists M_j \ (\text{word}_j, M_j) \in L_G \wedge L_\tau(w_{1j}) \sqsubseteq M_j$$

où word_j réfère au j^e mot de la phrase à analyser.

À présent, nous pouvons exprimer les relations de *pertinence* et *pertinence et satisfaction* sous forme de contraintes sur des arbres rectangulaires comme suit (pour des raisons de place, nous ne présentons que la contrainte d'accord, les autres se traitant de manière similaire).

Accord. Comme mentionné en Fig. 1, la propriété $S_0 : S_1 \rightsquigarrow S_2$ s'instancie sur des triplets de nœuds, ses instances I sont donc de la forme :

$$(S_0 : S_1 \rightsquigarrow S_2)@\langle w_{i_0j_0}, w_{i_1j_1}, w_{i_2j_2}\rangle$$

$\forall w_{i_0j_0}, w_{i_1j_1}, w_{i_2j_2} \in V$, tels que $w_{i_1j_1} \neq w_{i_2j_2}$. Une telle instance est pertinente ssi $w_{i_0j_0}$ est actif et possède une étiquette $L_\tau(w_{i_0j_0}) \sqsubseteq S_0^v$, et $w_{i_1j_1}$ et $w_{i_2j_2}$ sont des nœuds fils (relation de parenté notée $\downarrow$ ci-dessous) dont les étiquettes respectent les contraintes $L_\tau(w_{i_1j_1}) \sqsubseteq S_1^v$ et $L_\tau(w_{i_2j_2}) \sqsubseteq S_2^v$:

$$P(I) \Leftrightarrow \left(\begin{array}{c} w_{i_0j_0} \in V^+ \wedge w_{i_1j_1} \in \downarrow w_{i_0j_0} \wedge \\ w_{i_2j_2} \in \downarrow w_{i_0j_0} \wedge L_\tau(w_{i_0j_0}) \sqsubseteq S_0^v \wedge \\ L_\tau(w_{i_1j_1}) \sqsubseteq S_1^v \wedge L_\tau(w_{i_2j_2}) \sqsubseteq S_2^v \end{array} \right)$$

Sa satisfaction dépend du fait que les co-références définies dans les étiquettes de $w_{i_0j_0}, w_{i_1j_1}, w_{i_2j_2}$ soient satisfaites (contrainte (2) introduite précédemment). Donc, nous avons :

$$S(I) \Leftrightarrow \left[\begin{array}{l} P(I) \wedge \\ L_\tau(w_{i_0j_0}), L_\tau(w_{i_1j_1}), L_\tau(w_{i_2j_2}) \models E(S_0, S_1, S_2) \end{array} \right]$$

5 Implantation

La modélisation de l'analyse syntaxique des grammaires de propriété sous forme de CSP présentée ici repose sur la modélisation de Duchier *et al.* [2]. Dans leurs travaux, les auteurs proposaient une implantation en C++ au moyen de la bibliothèque Gecode [5].

L'extension de cette implantation aux propriétés à base de structures de traits nécessitait une refonte importante, en partie pour des raisons techniques liées au langage C++. Nous avons donc choisi de re-développer l'analyseur syntaxique en langage Python, toujours en utilisant Gecode pour la résolution de contraintes. [1]

Comme dans les travaux antérieurs de Duchier *et al.*, l'objectif principal de cette implantation n'est pas de fournir un analyseur performant dans le sens où il pourrait se comparer aux analyseurs existant pour d'autres formalismes grammaticaux, mais un analyseur dépourvu de toute heuristique, permettant ainsi au linguiste d'observer directement les conséquences de choix de représentations de la syntaxe de la langue. En quelque sorte, l'analyseur permet au linguiste d'avoir une estimation de la complexité d'un modèle de la langue.

Cette implantation étant en cours de développement, elle n'a pas pu être confrontée à des grammaires de taille réelle, nous ne disposons donc pas, à l'heure actuelle, de *benchmarks*. L'analyseur intègre actuellement un support expérimental pour les propriétés à base de traits, et une évaluation de son "efficacité" est envisagée, en utilisant la grammaire de [6].

1. Via un *binding* Gecode/Python développé en interne, et disponible librement à l'adresse `https://launchpad.net/pygecode`.

6 Conclusion

Dans cet article, nous avons présenté (i) une sémantique en théorie des modèles des Grammaires de Propriété à Traits, et (ii) la définition de l'analyse syntaxique en GP par conversion de cette sémantique en un problème de satisfaction de contraintes. L'intérêt de ce travail est de fournir une définition formelle des structures de traits dans le formalisme GP, et une implantation de ces structures de traits dans un analyseur de GP en programmation par contraintes (cette implantation est en cours de réalisation au moyen de la librairie Gecode).

Pour optimiser l'analyse syntaxique en pratique, nous envisageons plusieurs pistes de travail, notamment (i) la définition de l'analyse syntaxique par composition d'analyses partielles (chacune implantée sous forme d'un CSP), (ii) la parallélisation de l'exploration de l'arbre de recherche.

Enfin, une autre piste de recherche intéressante concerne la définition de poids à associer aux différentes contraintes en fonction des catégories impliquées ou du type de propriété (ordre entre catégories *vs* catégorie manquante) afin d'obtenir un modèle et un score de grammaticalité plus proche de l'intuition humaine lorsque des énoncés agrammaticaux sont manipulés.

Références

[1] Philippe Blache. Constraints, Linguistic Theories and Natural Language Processing. In D. Christodoulakis, editor, *Natural Language Processing*, Lecture Notes in Artificial Intelligence Vol. 1835. Springer-Verlag, 2000.

[2] Denys Duchier, Thi-Bich-Hanh Dao, Yannick Parmentier, and Willy Lesaint. Une modélisation en CSP des grammaires de propriétés. In *Sixièmes Journées Francophones de Programmation par Contraintes (JFPC)*, pages 123–132, Caen, France, 2010.

[3] Denys Duchier, Jean-Philippe Prost, and Thi-Bich-Hanh Dao. A model-theoretic framework for grammaticality judgements. In *Conference on Formal Grammar (FG2009)*, Bordeaux, France, 2009.

[4] Andrew Fall. *Reasoning with taxonomies*. PhD thesis, School of Computing Science, Simon Fraser University, December 1996.

[5] Gecode Team. Gecode : Generic constraint development environment, 2012. Available from `http ://www.gecode.org`.

[6] Jean-Philippe Prost. *Modelling Syntactic Gradience with Loose Constraint-based Parsing*. Cotutelle Ph.D. Thesis, Macquarie University, Sydney, Australia, and Université de Provence, Aix-en-Provence, France, December 2008.

Programmation par Contraintes sur les Séquences Infinies

Xavier Dupont[1] **Arnaud Lallouet**[1]

Y. C. Law[2] **J. H. M. Lee**[2] **C. F. K. Siu**[2]

[1]Université de Caen Basse-Normandie, 14000, Caen
[2]Université Chinoise de Hong-Kong
{xavier.dupont,arnaud.lallouet}@unicaen.fr
{yclaw,jlee,fksiu}@cse.cuhk.edu.hk

Abstract

La programmation par contraintes est habituellement utilisée pour résoudre des problèmes combinatoires pour lesquels l'ensemble des solutions est de cardinalité finie. Cette approche, bien adaptée aux problèmes d'ordonnancement, est moins applicable aux problèmes de planification, pour lesquels l'horizon doit être borné, et est complètement inapplicable à des problèmes temporels sans horizon. Nous proposons ici un cadre alternatif dans lequel les variables sont des séquences symboliques infinies. Il devient ainsi possible de spécifier des contraintes sur ces variables et de générer des solutions qui satisfont ces contraintes. Bien que chaque solution prise indépendamment soit de taille infinie, et que le nombre de solutions lui-même soit en général infini, il est possible de représenter l'ensemble des solutions de telle sorte qu'une solution arbitraire puisse être générée de façon efficace.

1 Introduction

La programmation par contraintes est un paradigme de programmation qui permet de résoudre des problèmes combinatoires de grandes tailles d'une manière générique. Pour résoudre un problème de programmation par contraintes, l'utilisateur spécifie les variables et les contraintes du problème et laisse le système trouver des solutions, c'est-à-dire des affectations de valeurs aux variables qui satisfont toutes les contraintes.

Cette approche est utilisée pour résoudre des problèmes d'allocation de ressources, de vérification de circuits électroniques, ou de raisonnement spatial et temporel. Certains types de problèmes, comme les problèmes d'ordonnancement, correspondent à des disciplines spécialisées de la programmation par contraintes, pour lesquelles des algorithmes de propagation spécifiques ont été développés [2]. En raison des succès de cette approche, la tendance est maintenant à l'extension du champ d'application de la programmation par contraintes à d'autres domaines, qui n'étaient pas habituellement couverts, comme la fouille de données [6] ou les logiques temporelles, qui sont le sujet de cet article.

Malgré la diversité des domaines d'application, il reste difficile de modéliser des problèmes dont la valeur des variables varie au cours du temps. Les solutions sont toujours des affectations de valeurs à un ensemble fini de variables, ce qui ne permet pas de modéliser l'évolution des valeurs d'une variable sur une durée arbitraire non définie à l'avance.

Ces caractéristiques ne posent pas beaucoup de problèmes pour la modélisation de problèmes d'ordonnancement, où l'objectif est en général de minimiser le temps nécessaire pour accomplir un ensemble de tâches dont une borne supérieure sur le temps nécessaire est connue à l'avance. La situation est plus délicate pour les problèmes de planification, pour lesquels le but est de minimiser le temps nécessaire pour atteindre une configuration désirable, et ce sans aucune information connue a priori concernant ce temps. Des méthodes spécifiques ont été élaborées pour la résolution de ce type de problème, et les approches basées sur la programmation par contraintes utilisent en général une résolution itérative, qui consiste à allonger le temps imparti jusqu'à ce qu'une solution soit trouvée [5]. Les ressources nécessaires à la résolution augmentent au

fur et à mesure des résolutions infructueuses, ce qui rend parfois l'approche inapplicable.

Enfin, les domaines de variables existants ne permettent pas de modéliser des problèmes temporels sans horizon, pour lesquels l'objectif est simplement de satisfaire des contraintes temporelles, et ce de façon indéfinie sur la durée. Une solution sera alors une séquence infinie d'affectations, qui devra à tout instant pouvoir être étendue de façon à continuer de satisfaire toutes les contraintes.

Des formalismes, parfois très généraux [9], ont été développés pour modéliser ce type de problème. Mais ces approches fonctionnent toutes par extension de l'horizon temporel, et font penser aux méthodes utilisées pour la résolution de problèmes de planification avec des CSPs.

Nous présentons dans cet article un formalisme pour traiter des problèmes temporels en utilisant les techniques de la programmation par contraintes. Les applications couvrent des domaines qui étaient jusqu'à présent exclus du domaine pour les raisons évoquées. Des exemples sont la vérification de systèmes interactifs ou dynamiques, la planification sans horizon, et l'ajustement optimal de systèmes réactifs à un environnement changeant. Des applications particulières peuvent être l'ajustement en continu d'une chaîne de production à certaines conditions, telles que la disponibilité des produits, ou bien, dans un domaine plus ludique, la génération en continue de musique à partir d'un ensemble de règles d'harmonisation.

2 Problème de satisfaction de contraintes

Un problème de satisfaction de contraintes (CSP), au sens large, est un tuple $\mathcal{S} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ où $\mathcal{X}$ est un ensemble fini de variables, $\mathcal{D}$ est l'ensemble des domaines associés aux variables, et $\mathcal{C}$ est un ensemble fini de contraintes. Pour chaque variable $X \in \mathcal{X}$, on note $D(X) \in \mathcal{D}$ le domaine associé à X.

Une contrainte $C \in \mathcal{C}$ peut être vue comme l'application d'une relation R d'arité a, notée $R(C)$, sur un tuple $\tau = (\tau[1], \tau[2], \ldots, \tau[a])$ de a variables, appelé portée de C et notée $\pi(C)$, où R est un sous-ensemble du produit cartésien des domaines des variables de τ, c'est-à-dire $R \subseteq D(\tau)$ avec $D(\tau) = D(\tau[1]) \times D(\tau[2]) \times \cdots \times D(\tau[a])$. La contrainte est satisfaite par une affectation $\alpha \in D(\tau)$ si $\alpha \in R$.

Une solution s d'un CSP dont les variables sont $\mathcal{X} = (X_1, X_2, \ldots, X_n)$ est l'affectation d'une valeur à chaque variable, où chaque valeur est prise dans le domaine de la variable correspondante, et de telle sorte que toutes les contraintes sont satisfaites, c'est-à-dire que $s \in D(X_1) \times D(X_2) \times \cdots \times D(X_n)$ et

$$\forall C \in \mathcal{C}, s_{|\pi(C)} \in R(C)$$

où pour une contrainte C et une affectation s, $s_{|\pi(C)}$ dénote la projection de s sur la portée de C, c'est-à-dire l'affectation partielle qui à chaque variable de la portée de C associe la valeur correspondante de s.

L'objectif est de trouver la ou les solutions qui satisfont toutes les contraintes.

En général, les domaines des variables sont de cardinalités finies, ce qui a pour conséquence que le nombre de solutions possibles et la taille de l'espace de recherche sont finis. Dans ces conditions, l'espace de recherche peut toujours être parcouru intégralement. L'objectif est donc de rendre la recherche de solutions aussi rapide que possible en identifiant des régions infructueuses dont on pourra éviter l'exploration. Cela se fait à l'aide de propagateurs qui permettent d'exploiter la sémantique des contraintes afin de réduire le domaine des variables et choisir des valeurs intéressantes pour la suite de la recherche.

Ces caractéristiques des CSPs ne sont pas présentes dans les StCSPs. En effet, l'espace de recherche n'est pas de taille finie, et le nombre de solutions possibles est en général infini. Cependant, pour certains types de contraintes, il est possible de représenter l'ensemble des solutions par une structure de données de taille finie, qui permet de générer n'importe quelle solution de façon efficace, et cette structure peut être calculée par résolution d'un CSP.

3 StCSPs

Dans un problème de Satisfaction de Contraintes sur Flux (StCSP), de l'anglais "Stream Constraint Satisfaction Problem", les domaines des variables sont des ensembles de flux, et les contraintes sont appliquées sur des termes. Comme la notion de contrainte est trop générale, nous considérons seulement des contraintes dites "de voisinage". Ces notions sont détaillées dans ce qui suit.

3.1 Flux

Les StCSPs sont des CSPs dont les domaines des variables sont des ensembles de flux. Un *flux* α est une séquence ordonnée $\langle \alpha(0), \alpha(1), \ldots \rangle$ de datons, où chaque daton $\alpha(i)$ appartient à un ensemble fini commun Σ_α, l'alphabet associé à α. On note $\alpha(i, j)$, $0 \leq i \leq j$ la séquence finie $\langle \alpha(i), \alpha(i+1), \ldots, \alpha(j) \rangle$. $\alpha(i, \infty)$ correspond à la séquence infinie $\langle \alpha(i), \alpha(i+1), \ldots \rangle$, et $\alpha(i, i)$ correspond à $\langle \alpha(i) \rangle$, qui est identifiable à $\alpha(i)$.

Nous définissons des opérateurs sur les flux qui permettent de symboliser des parties d'un flux.

L'opérateur "Next" associe à un flux le même flux amputé de son premier daton. Pour un flux $\alpha = \alpha(0, \infty)$, $\text{Next}(\alpha) = \alpha(1, \infty)$.

L'opérateur "Next" doit être considéré comme un opérateur de projection. Le flux renvoyé par "Next" est intrinsèquement le même que le flux d'entrée. En particulier, les datons oubliés par "Next" restent définis. L'opérateur "Next" sert donc à calculer différentes vues d'un même flux.

Cet opérateur peut être composé. Nous notons $\text{Next}^n(\alpha)$ le flux $\text{Next}(\text{Next}^{n-1}(\alpha))$, avec $\text{Next}^0(\alpha) = \alpha$.

Les flux peuvent être représentés à l'aide de l'opérateur $(\cdot)^\omega$. On note a^ω le flux $\langle a, a, \ldots \rangle$, d'alphabet $\Sigma = \{a\}$. Cette notation peut être étendue aux mots finis. Si $w = \langle w[1], w[2], \ldots, w[\ell] \rangle$ est un mot de longueur ℓ, w^ω représente le flux $\langle w[0], w[1], \ldots, w[\ell], w[0], w[1], \ldots, w[\ell], \ldots \rangle$.

Dans le cas où S est un ensemble de mots finis, S^ω représente l'ensemble des mots infinis qui peuvent être construits avec des mots de S. Une définition récursive est $S^\omega = \{ww' \mid w \in S \wedge w' \in S^\omega\}$. En particulier, l'ensemble des mots infinis sur un alphabet fini Σ est noté Σ^ω.

Un exemple de flux est le flux $\alpha = (abc)^\omega = \langle a, b, c, a, b, c, \ldots \rangle$. Pour ce flux, $\text{Next}^2(\alpha) = \langle c, a, b, c, a, b, \ldots \rangle$ et $\text{Next}^3(\alpha) = \alpha$.

3.2 Variables flux

Cette section introduit le type de variable utilisé dans les StCSPs, que nous appelons *variable flux*. Le domaine d'une variable flux est un ensemble de flux, potentiellement infini. Le domaine initial d'une variable flux est l'ensemble des séquences qui peuvent être construites sur un alphabet fini donné. Si X est une variable flux et $d(X)$ est l'alphabet associé à X, le domaine de X est $D(X) = d(X)^\omega$.

L'opérateurs "Next" est étendu aux domaines de variables flux. Pour une variable flux X de domaine $D(X)$, $\text{Next}(D(X)) = \{\text{Next}(\alpha) \mid \alpha \in D(X)\}$.

Les variables flux sont identifiées avec leurs domaines. Pour une variable flux X, $\text{Next}(X) = \text{Next}(D(X))$.

3.3 Termes

Un *terme* est construit par application de l'opérateur "Next" sur une variable flux. L'ensemble des termes peut être construit inductivement :

- une variable flux est un terme
- si t est un terme, alors $\text{Next}(t)$ est un terme

La notion de domaine est étendue aux termes. Pour une variable flux, le domaine du terme est simplement le domaine de la variable. Sinon, t est de la forme $\text{Next}(t')$, où t' est un terme, et $D(t) = D(\text{Next}(t')) = \{\text{Next}(\alpha) \mid \alpha \in D(t')\}$.

Un terme peut toujours être écrit sous la forme $\text{Next}^n(X)$, où X est une variable flux et $n \in \mathbb{N}$. Par convention, $\text{Next}^0(X) = X$ et $\text{Next}^n(X) = \text{Next}(\text{Next}^{n-1}(X))$ si $n \neq 0$. Le domaine d'un terme est donc $D(t) = D(\text{Next}^n(X)) = \{\text{Next}^n(\alpha) \mid \alpha \in D(X)\}$.

Exemple Soit X une variable flux de domaine $D(X) = a(b)^\omega$. Alors $\text{Next}(X)$ est un terme, dont le domaine est $D(\text{Next}(X)) = b^\omega$ et $D(\text{Next}^n(X)) = D(\text{Next}(X)) = b^\omega$ pour tout n strictement positif.

3.4 Contraintes

Pour la formalisation des contraintes sur les StCSPs, nous réutilisons les notations de la section 2.

Une contrainte est l'application d'une relation R d'arité a sur un tuple τ de a termes. Pour une contrainte C de portée $\pi(C) = (t_1, t_2, \ldots, t_n)$, la relation $R(C)$ associée à C décrit un sous-ensemble du produit cartésien des domaines des variables de sa portée, c'est-à-dire $R(C) \subseteq D(t_1) \times D(t_2) \times \cdots \times D(t_n)$.

Une affectation $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in D(t_1) \times D(t_2) \times \cdots \times D(t_n)$ satisfait C si $\alpha \in R_C$.

Cette formulation est très générale. Un exemple de contrainte est $C : R_C(t_1, t_2)$ où t_1 et t_2 sont deux termes de domaines $D(t_1) = D(t_2) = \{a, b\}^\omega$, et $R_C = \{(\alpha_1, \alpha_2) \in D(t_1) \times D(t_2) \mid \forall n \in \mathbb{N}, |\alpha_1(0, n)|_a < |\alpha_2(0, n)|_a\}$, où $|w|_a$ représente le nombre de a présents dans $w \in \{a, b\}^*$. Cette contrainte impose que tout préfixe fini de α_1 contienne moins de a que le préfixe de α_2 de même longueur. L'affectation $\alpha = (bbb\ldots, aaa\ldots)$ satisfait cette contrainte, car le nombre de a dans n'importe quel préfixe de $bbb\ldots$ est toujours nul. Il est cependant difficile de représenter l'ensemble des affectations qui satisfont C autrement que par la définition de la contrainte elle-même. Il s'agit d'une contrainte globale au sens des flux, c'est-à-dire qui porte sur l'ensemble des datons de chaque terme.

Contraintes de voisinage

Nous définissons la notion de contrainte de voisinage. Une contrainte de voisinage s'applique localement sur les flux qui sont dans sa portée. Il s'agit toujours de l'application d'une relation n-aire sur un tuple de n termes, à la différence que la relation décrit un sous-ensemble du produit cartésien des alphabets des domaines des variables de sa portée, c'est-à-dire que pour une contrainte de voisinage $C : R_C(t_1, t_2, \ldots, t_n)$, $R_C \subseteq d(t_1) \times d(t_2) \times \cdots \times d(t_n)$.

Il est impossible de contraindre l'ensemble des datons des flux avec un nombre fini de contraintes de voisinage. C'est pourquoi nous définissons un mécanisme de réplication pour ce type de contrainte, qui permet de contraindre l'ensemble des datons d'une manière structurée.

Le plus simple est de considérer qu'une contrainte de voisinage est satisfaite si et seulement si elle est satisfaite en tout point du système de flux. Formellement, une contrainte de voisinage $C : R_C(t_1, t_2, \ldots, t_n)$ est satisfaite par une affectation $(\alpha_1, \alpha_2, \ldots, \alpha_n) \in D(t_1) \times D(t_2) \times \cdots \times D(t_n)$ si et seulement si :

$$\forall i \geq 0, (\alpha_1[i], \alpha_2[i], \ldots, \alpha_n[i]) \in R_C$$

Exemple Soient X et Y deux variables flux, dont les domaines sont $D(X) = D(Y) = \{a, b\}^\omega$, et $C_E : R_E(X, Y, \text{Next}(X))$ une contrainte de voisinage sur X et Y, qui impose sur Y la lettre la plus fréquente dans X et $\text{Next}(X)$. La table de R_E est donnée dans la figure 1. Dans le cas où les deux lettres sont différentes, chacune peut être considérée comme la plus fréquente. Le réseau de contraintes correspondant à l'application de C_E sur X, Y et $\text{Next}(X)$ est schématisé dans la figure 2. Dans cette contrainte, la valeur courante de Y dépend de la valeur future de X à tout instant.

Un exemple de solution est $\sigma = (\sigma(X), \sigma(Y))$, où $\sigma(X) = (aab)^\omega$ et $\sigma(Y) = (abbaba)^\omega$. En revanche, des flux dont les débuts sont respectivement $\sigma(X) = \langle a, a, b, b, a, b, \ldots \rangle$ et $\sigma(Y) = \langle b, b, b, b, b, b, b, \ldots \rangle$ ne peuvent pas satisfaire la contrainte, puisque le flux $\sigma(Y)$ doit commencer par un a étant donné $\sigma(X)$.

3.5 StCSPs

Un problème de Satisfaction de Contraintes sur Flux (StCSP) $\mathcal{S} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est la donnée d'un ensemble de variables flux $\mathcal{X} = (X_1, X_2, \ldots, X_n)$, d'un ensemble de domaines $\mathcal{D}$ et d'un ensemble de contraintes $\mathcal{C}$ sur ces variables. Le domaine d'une variable $X \in \mathcal{X}$ est noté $D(X)$. Il s'agit de l'ensemble des séquences infinies qui peuvent être construites sur l'alphabet $d(X)$ associé à la variable X, c'est-à-dire $D(X) = d(X)^\omega$.

R_E		
a	a	a
a	a	b
a	b	b
b	a	a
b	b	a
b	b	b

FIGURE 1 – Relation associée à l'exemple de contrainte de voisinage C_E

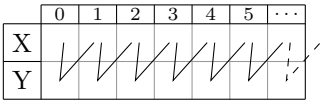


FIGURE 2 – Réseau de contraintes correspondant à l'application de C_E

$\mathcal{C}$ est un ensemble fini de contraintes de voisinage sur des termes. Les variables, les termes et les contraintes de voisinage ont été décrits dans les sections précédentes.

Une solution à un StCSP est une affectation $(\sigma_1, \sigma_2, \ldots, \sigma_n) \in \mathcal{D}(X_1) \times D(X_2) \times \cdots \times \mathcal{D}(X_n)$ qui satisfait toutes les contraintes.

La *temporalité* d'un StCSP est définie comme étant le plus grand nombre d'itérations de l'opérateur "Next" parmi tous les termes figurant dans le problème. Ainsi, un StCSP de temporalité 1 ne contient que des termes de la forme X ou $\text{Next}(X)$, où X est une variable flux. Dans ce qui suit, un k-StCSP désigne un StCSP de temporalité k.

Une variable est un terme de temporalité 0. Un exemple de terme de temporalité 3 est $\text{Next}^3(X)$, où X est une variable flux. Enfin, $C : R_C(X, \text{Next}^2(X), \text{Next}(Y))$ est un exemple de contrainte de temporalité 2, où R_C décrit un sous-ensemble arbitraire de $d(X) \times d(X) \times d(Y)$.

4 Représentation de l'ensemble des solutions

Il est possible de représenter l'ensemble des solutions d'un StCSPs par une structure de taille finie, qui peut être calculée par un solveur de CSPs tel que Gecode. Nous présentons la méthode pour les StCSPs de temporalité 1. Nous présenterons dans la section suivante une méthode qui permet de transformer les StCSPs de temporalité plus grande en StCSP de temporalité 1.

Les contraintes d'un 1-StCSP peuvent être de trois formes :

Contraintes point-à-point Une contrainte point-à-point est une contrainte dans laquelle tous les termes sont de temporalité 0, c'est-à-dire qu'elle n'utilise pas l'opérateur "Next". Il s'agit d'une contrainte sur le produit cartésien des alphabets des variables apparaissant dans la contraintes. L'ensemble des contraintes de temporalité 0 représente un CSP classique dont toutes les solutions peuvent être calculées selon les méthodes habituelles. Ce type de contrainte permet de définir, par exemple, les états autorisés d'un système réactif.

Contraintes mixtes Une contrainte mixte est une contrainte dans laquelle certains termes sont de temporalité 0, et d'autres sont de temporalité 1. Il s'agit d'une contrainte de transition, car elle connecte la valeur courante des variables apparaissant dans les termes de temporalité 0, avec la valeur suivante des variables apparaissant dans les termes de temporalité 1. Ce type de contrainte permet de spécifier, par exemple, les règles d'évolution d'un automate.

Contraintes singulières Les contraintes singulières sont des contraintes dont aucun terme n'est de temporalité 0. Dans le cas des 1-StCSPs, si aucun terme n'est de temporalité 0, alors tous les termes sont de temporalité 1. Une telle contrainte correspond à une contrainte point-à-point qui s'applique partout sauf au premier point du système de flux, d'où le nom.

4.1 Système de transitions associé à un 1-StCSP

Un 1-StCSP qui ne contient que des contraintes de voisinage peut être transformé en un système de transitions.

Un système de transitions $\mathcal{T} = (Q, \delta)$ est un ensemble d'états Q accompagné d'une relation de transition $\delta \subseteq Q \times Q$. Un chemin dans $\mathcal{T}$ est une séquence $\langle q_1, q_2, \ldots, q_n \rangle$ de n états tels que $(q_i, q_{i+1}) \in \delta$ pour $0 < i \leq n$. La définition s'étend de façon naturelle aux chemins de longueurs infinies.

Pour présenter la transformation d'un StCSP en système de transitions, quelques définitions et notations sont nécessaires. Soit $\mathcal{S} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ un StCSP où :

- $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ est un ensemble de n variables flux
- $\mathcal{D} = \{d(X_1)^\omega, d(X_2)^\omega, \ldots, d(X_n)^\omega\}$ est l'ensemble des domaines associés aux variables, où $d(X_i)$ est l'alphabet associé à la variable flux X_i
- $\mathcal{C} = \mathcal{C}^0 \cup \mathcal{C}^1$
- $\mathcal{C}^0 = \{C_1^0, C_2^0, \ldots, C_{k_0}^0\}$ est l'ensemble des k_0 contraintes point-à-point de $\mathcal{S}$
- $\mathcal{C}^1 = \{C_1^1, C_2^1, \ldots, C_{k_1}^1\}$ est l'ensemble des k_1 contraintes mixtes ou singulières de $\mathcal{S}$

Une solution de $\mathcal{S}$ est une affectation $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in d(X_1)^\omega \times d(X_2)^\omega \times \cdots \times d(X_n)^\omega$ qui satisfait toutes les contraintes, et peut être vue comme une séquence infinie de n-tuples $\tau \in d(X_1) \times d(X_2) \times \cdots \times d(X_n)$. Nous notons $\tau[\alpha]$ la séquence de tuples associée à α. Elle peut être définie récursivement en utilisant l'opérateur "Next" :

$$\tau[\alpha] = (\alpha_1(0), \alpha_2(0), \ldots, \alpha_n(0))\tau[\text{Next}(\alpha)]$$

Une solution de $\mathcal{S}$ correspond alors à la trace d'une exécution d'un système de transitions $\mathcal{T}(\mathcal{S}) = (Q, \delta)$, pour lequel $Q \subseteq d(X_1) \times d(X_2) \times \cdots \times d(X_n)$, et $(q_1, q_2) \in \delta$ si et seulement si il existe une solution α et un facteur $\alpha(i, i+1)$ de α tel que $q_1 = \tau[\alpha](i)$ et $q_2 = \tau[\alpha](i+1)$.

L'ensemble des états est l'ensemble des combinaisons qui satisfont toutes les contraintes point-à-point :

$$Q = \{\tau \in d(X_1) \times \cdots \times d(X_n) \mid \forall C \in \mathcal{C}^0, \tau \vDash C\}$$

où $\tau \vDash C$ signifie que l'affectation τ satisfait C.

La relation de transition est construite avec les contraintes mixtes :

$$\delta = \{(\tau_0, \tau_1) \in Q \times Q \mid \forall C \in \mathcal{C}^1, (\tau_0, \tau_1) \vDash C\}$$

où $(\tau_0, \tau_1) \vDash C$ signifie qu'un couple de tuples satisfait C, c'est-à-dire que les termes de temporalité 0 correspondent à des valeurs prises dans τ_0 et les termes de temporalité 1 correspondent à des valeurs prises dans τ_1.

Il s'ensuit que toutes les traces d'exécution du système de transitions $\mathcal{T}(\mathcal{S})$ associé à un StCSP $\mathcal{S}$ de temporalité 1 correspondent à des solutions de $\mathcal{S}$, car, par construction, les contraintes point-à-point et les contraintes mixtes de $\mathcal{S}$ sont satisfaites.

4.2 Calcul de l'ensemble des solutions

Nous montrons dans cette section que l'ensemble des solutions d'un 1-StCSP $\mathcal{S}$ ne comportant que des contraintes de voisinage, c'est à dire le système de transition $\mathcal{T}(\mathcal{S})$ associé à $\mathcal{S}$, peut être modélisé par un problème de satisfaction de contraintes, et donc calculé par un solveur de CSP.

Soit $\mathcal{S} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ un 1-StCSP défini comme précédemment.

Le CSP $\mathcal{P}(\mathcal{S})$ associé à $\mathcal{S}$ comporte $2n$ variables X_1^0, $\ldots$, X_n^0, $X_1^1, \ldots, X_n^1$ où les domaines des variables sont $D(X_i^0) = D(X_i^1) = d_i$ et représentent respectivement la valeur de X_i à l'instant courant et la valeur de X_i à l'instant suivant.

Les contraintes point-à-point sont appliquées sur l'instant courant et l'instant suivant. Pour chaque contrainte $C_i(X_{j_1}, \ldots, X_{j_a}) \in \mathcal{C}^0$ d'arité a, les contraintes correspondantes dans $\mathcal{P}(\mathcal{S})$ sont $C_i(X^0_{j_1}, \ldots, X^0_{j_a})$ et $C_i(X^1_{j_1}, \ldots, X^1_{j_a})$.

Les contraintes mixtes sont appliquées partiellement à l'instant courant et partiellement à l'instant suivant. Les termes de temporalité 0 correspondent aux variables de l'instant courant, et les termes de temporalité 1 correspondent aux variables de l'instant suivant. Une contrainte mixte $C_i(\text{Next}^{b_1}(X_{j_1}), \ldots, \text{Next}^{b_a}(X_{j_a})) \in \mathcal{C}^1$ d'arité a dans $\mathcal{S}$ correspond à la contrainte $C_i(X^{b_1}_{j_1}, \ldots, X^{b_a}_{j_a})$ dans le CSP, où $b_k \in \{0, 1\}$ correspond à la temporalité du terme correspondant (1 si l'opérateur "Next" est utilisé et 0 sinon).

Les solutions de $\mathcal{P}(\mathcal{S})$ correspondent à l'ensemble des transitions de $\mathcal{T}(\mathcal{S})$ car les contraintes point-à-point sont satisfaites à chaque instant, et les contraintes mixtes sont satisfaites pour chaque transition. Il est donc possible de calculer le système de transitions $\mathcal{T}(\mathcal{S})$ associé à un 1-StCSP $\mathcal{S}$ en résolvant le problème de satisfaction de contraintes $\mathcal{P}(S)$ correspondant.

5 Réduction de temporalité

Cette section décrit comment un StCSP de temporalité supérieure à 1 peut être transformé en un 1-StCSP équivalent. Le principe consiste à ajouter de la mémoire au StCSP, en utilisant des variables auxiliaires qui contiennent les valeurs des variables aux instants futurs. Ces variables sont reliées entre elles par des contraintes d'égalité. De cette façon, à chaque instant, le système peut contenir de l'information sur les valeurs futures de certaines de ses variables.

Soient $\mathcal{S}$ un k-StCSP et $\mathcal{S}_R$ le 1-StCSP équivalent. $\mathcal{S}_R$ peut être calculé par itération de la procédure suivante :

Soit $\text{Next}^k(X)$ un terme de temporalité k. Soient X^{k-1} une nouvelle variable de domaine $D(X^{k-1}) = D(X)$, et $X^{k-1} = \text{Next}^{k-1}(X)$ une nouvelle contrainte d'égalité. Alors toutes les occurences du terme $\text{Next}^k(X)$ peuvent être remplacées par $\text{Next}(X^{k-1})$, qui est de temporalité 1. La contrainte d'égalité est de temporalité $k-1$. Si X était la seule variable de temporalité k dans le problème, alors la temporalité du problème a effectivement été réduite de 1. Sinon, la procédure doit être appliquée une nouvelle fois sur une variable différente. Comme le nombre de variables est fini, cette procédure finira par produire un StCSP de temporalité $k-1$, et finalement un StCSP de temporalité 1. La section précédente pourra alors être appliquée pour calculer l'ensemble des solutions.

Les StCSPs $\mathcal{S}$ et $\mathcal{S}_R$ sont équivalents au sens où les solutions de $\mathcal{S}_R$ sont identiques aux solutions de $\mathcal{S}$ lorsqu'elles sont projetées sur les variables de $\mathcal{S}$. En effet, les variables auxiliaires représentent des variables déjà présentes dans $\mathcal{S}$, mais à des instants différents. Dès lors qu'un solution est trouvée, les valeurs de ces variables deviennent redondantes et peuvent être éliminées de la représentation d'une solution. En revanche, ces variables sont nécessaires à la représentation de l'ensemble des solutions d'un k-StCSP.

5.1 Système de transitions étiqueté associé à un k-StCSP

Il est naturellement possible de représenter un k-StCSP $\mathcal{S}$ par le système de transitions associé au StCSP réduit $\mathcal{T}(\mathcal{S}_R)$. Cependant, ceci implique de représenter les variables auxiliaires de façon explicite, alors qu'elles ne sont qu'un artefact de la procédure utilisée pour obtenir une représentation finie de l'ensemble des solutions de $\mathcal{S}$. Les systèmes de transitions étiquetés permettent de représenter l'ensemble des solutions d'un k-StCSP d'une manière plus fidèle à sa spécification.

Une système de transitions étiqueté est un tuple $\mathcal{T}_E = (Q, \delta, E, L)$ où :
- Q est un ensemble d'états
- $\delta \subseteq Q \times Q$ est la relation de transition
- E est un ensemble d'étiquettes
- $L : Q \to E$ est une fonction d'étiquetage qui associe à chaque état une étiquette

Soit $\mathcal{T}_E(\mathcal{S})$ le système de transitions étiqueté associé à $\mathcal{S}$. Les états et la relation de $\mathcal{T}_E(\mathcal{S})$ sont exactement ceux de $\mathcal{T}(\mathcal{S}_R)$.

Les étiquettes correspondent à la projection de Q sur l'ensemble des variables de $\mathcal{S}$, et la fonction d'étiquetage effectue cette opération. Plus précisément, soit $q = (a_1, \ldots, a_n, a_{n+1}, \ldots, a_{n+m}) \in Q$, avec $(a_1, \ldots, a_n) \in d(X_1) \times \cdots \times d(X_n)$, où X_1, …, X_n sont les variables de $\mathcal{S}$, et $(a_{n+1}, \ldots, a_{n+m}) \in d(X_{n+1}) \times \cdots \times d(X_{n+m})$, où X_{n+1}, …, X_{n+m} sont des variables auxiliaires issues de la procédure de réduction de temporalité. Alors $L(q) = (a_1, \ldots, a_n)$.

6 Exemples

Nous présentons deux exemples de modélisation par des StCSPs. Le premier est un exemple très petit dont le thème est la synchronisation de feux de circulation

R_C	
V	O
V	R
O	V
R	V
O	O
R	R

R_{Ev}	
V	O
O	R
R	R
R	V

FIGURE 3 – Tables des relations utilisées par les contraintes de $\mathcal{S}_E$

sur un carrefour, qui permet d'expliciter les notions qui ont été introduites dans les sections précédentes. Le deuxième exemple est une adaptation du problème de configuration d'une chaîne de production automobile, qui est décrit dans CSPLib.

6.1 Carrefour

Cet exemple simplifié permet d'illustrer les applications des StCSPs pour la modélisation de problèmes temporels. Il s'agit de modéliser l'évolution de feux de circulation à un carrefour. Le problème $\mathcal{S}_E = \{\mathcal{X}_E, \mathcal{D}_E, \mathcal{C}_E\}$ comprend deux variables $\mathcal{X}_E = \{X, Y\}$ de domaines $D(X) = D(Y) = \{V, O, R\}$, qui représentent les états de deux feux de signalisation, où V symbolise un feu vert, O symbolise un feu orange et R symbolise un feu rouge. Les contraintes permettent de modéliser les états autorisés pour le système, ainsi que, de façon indépendante, les évolutions possibles pour chaque feu, dont la couleur évolue au cours du temps.

Nous définissons une relation R_C, dite "relation de compatibilité", pour spécifier les états autorisés du système, dont la table est donnée dans la figure 3. Cette relation permet de définir une contrainte point-à-point sur les feux. Dans le cas où le carrefour utiliserait plusieurs feux, il suffirait d'appliquer la relation sur chaque paire de feux pris deux à deux. Dans cet exemple, il n'y a que deux feux, donc une seule contrainte est nécessaire :

$$C_C : R_C(X, Y)$$

Comme il est pratique d'utiliser des tables pour représenter les relations, nous distinguons la relation de la contrainte elle-même, qui représente l'application de sa relation à un tuple de termes. Ici, la contrainte C_C représente l'applications de la relation R_C aux termes X et Y.

Les évolutions des feux sont spécifiées par la relation R_{Ev}, qui permet de définir des contraintes mixtes

X^0	Y^0	X^1	Y^1
V	R	O	R
O	R	R	V
O	R	R	R
R	V	R	O
R	O	V	R
R	O	R	R
R	R	V	R
R	R	R	V
R	R	R	R

FIGURE 4 – Solutions de $\mathcal{P}(\mathcal{S}_E)$

E_X et E_Y qui décrivent l'évolution des feux X et Y respectivement.

$$E_X : R_{Ev}(X, \text{Next}(X))$$

$$E_Y : R_{Ev}(Y, \text{Next}(Y))$$

CSP associé à $\mathcal{S}_E$ Le problème est de temporalité 1, il n'est donc pas nécessaire d'appliquer la procédure de réduction de temporalité, et le CSP associé à $\mathcal{S}_E$ est une traduction directe de $\mathcal{S}_E$ en CSP. Aucune variable n'est ajoutée, donc les variables du CSP sont $\mathcal{X} = \{X^0, Y^0, X^1, Y^1\}$. Les domaines sont $D(X^0) = D(X^1) = D(Y^0) = D(Y^1) = \{V, O, R\}$.

Les contraintes point-à-point sont appliquées sur chaque groupe, ce qui donne deux contraintes pour la contrainte de compatibilité C_C :

$$C_0 : R_C(X^0, Y^0)$$

$$C_1 : R_C(X^1, Y^1)$$

Les contraintes mixtes sont traduites directement.

$$E_X : R_{Ev}(X^0, X^1)$$

$$E_Y : R_{Ev}(Y^0, Y^1)$$

L'ensemble des solutions de ce CSP est donné par la figure 4, et correspondent au système de transitions de la figure 5.

Tous les chemins de $\mathcal{T}(\mathcal{S}_E)$ correspondent à une solution de $\mathcal{S}$. Un exemple de début de solution est :

X	V	O	R	R	V	O	R	...
Y	R	R	V	O	R	R	R	...

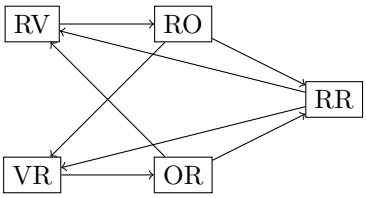


FIGURE 5 – Système de transitions $\mathcal{T}(\mathcal{S}_E)$ associé à $\mathcal{S}_E$

c_1	o_1	$\neg o_2$	o_3	o_4	$\neg o_5$
c_2	$\neg o_1$	$\neg o_2$	$\neg o_3$	o_4	$\neg o_5$
c_3	$\neg o_1$	o_2	$\neg o_3$	$\neg o_4$	o_5
c_4	$\neg o_1$	o_2	$\neg o_3$	o_4	$\neg o_5$
c_5	o_1	$\neg o_2$	o_3	$\neg o_4$	$\neg o_5$
c_6	o_1	o_2	$\neg o_3$	$\neg o_4$	$\neg o_5$

FIGURE 6 – Classes et options utilisées pour $\mathcal{S}_V$

6.2 Production automobile

Le problème de configuration d'une chaîne de production automobile (problème n°1 de CSPLib) est un problème classique de satisfaction de contraintes. Nous présentons une adaptation de ce problème au cadre des StCSPs.

Dans ce problème, une séquence infinie de voitures doit être produite, de telle sorte que les machines utilisées pour l'installation des options ne soient jamais surchargées. Les voitures peuvent être de différentes classes, en fonction des options qu'elles utilisent. Les options sont modélisées par des variables binaires, indépendantes les unes des autres.

Les options sont installées par des machines appropriées. Certaines options sont plus difficiles à mettre en place que d'autres, ce qui fait que pour un nombre donné de voitures, peu de voitures pourront recevoir une option difficile à installer. Cette notion de difficulté est encodé par une capacité k/n, qui exprime que pour n voitures consécutives, au plus k voitures pourront recevoir l'option.

Nous considérons une instanciation du problème avec 6 classes de voitures $\{c_1, c_2, \ldots, c_6\}$ qui sont des combinaisons de 5 options $\{o_1, o_2, \ldots, o_5\}$. La table est données sur la figure 6, ou o_k signifie que l'option o_k est installée, et $\neg o_k$ signifie que que l'option n'est pas installée pour la classe correspondante.

Les contraintes de capacités pour chaque option sont données dans la table 7.

Modélisation Nous notons $\mathcal{S}_V = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ la modélisation StCSP du problème présenté dans les paragraphes précédents. Nous utilisons cinq variables flux

o_1	o_2	o_3	o_4	o_5
1/2	2/3	1/3	2/5	1/5

FIGURE 7 – Contraintes de capacité pour les options de $\mathcal{S}_V$

O_1, O_2, O_3, O_4 et O_5, de domaines $D(O_1) = D(O_2) = D(O_3) = D(O_4) = D(O_5) = \{0,1\}^\omega$.

À tout instant, la combinaison des valeurs courantes des options doit correspondre à une des classes définies. Il s'agit d'une contrainte extensionnelle qui représente la figure 6.

$$
\begin{aligned}
(O_1, O_2, O_3, O_4, O_5) \in \{ \\
&(1,0,1,1,0), \\
&(0,0,0,1,0), \\
&(0,1,0,0,1), \\
&(0,1,0,1,0), \\
&(1,0,1,0,0), \\
&(1,1,0,0,0) \\
\}
\end{aligned}
$$

En ce qui concerne les contraintes de capacité, il est possible d'exploiter les contraintes habituellement utilisées en programmation par contraintes. Les cinq contraintes suivantes permettent d'exprimer les containtes de capacité.

$$
\begin{aligned}
&O_1 + \text{Next}(O_1) \leq 1 \\
&O_2 + \text{Next}(O_2) + \text{Next}^2(O_2) \leq 2 \\
&O_3 + \text{Next}(O_3) + \text{Next}^2(O_3) \leq 1 \\
&O_4 + \text{Next}(O_4) + \text{Next}^2(O_4) + \\
&\qquad \text{Next}^3(O_4) + \text{Next}^4(O_4) \leq 2 \\
&O_5 + \text{Next}(O_5) + \text{Next}^2(O_5) + \\
&\qquad \text{Next}^3(O_5) + \text{Next}^4(O_5) \leq 1
\end{aligned}
$$

Ce problème est de temporalité 4. Après réduction de temporalité, le problème contient 8 variables auxiliaires, ce qui génère un CSP de 26 variables. Le systèmes de transition correspondant est suffisamment grand pour ne pas être inclus dans la page. Cependant, les systèmes générés par ce problème et ses variantes ne sont en général pas très grands comparés à ce qu'il est possible de manipuler avec des systèmes informatisés. Ceci rend ce problème particulièrement approprié à l'étude des StCSPs.

7 Recherche incrémentale

Il n'est pas nécessaire de calculer le système de transitions entièrement pour trouver une solution. La modélisation sous forme de CSP peut être adaptée de telle

Algorithme : Recherche incrémentale

$H \leftarrow \emptyset$
Pour chaque solution (τ_1, τ_2) de $\mathcal{P}$
  ajouter τ_1 et τ_2 à H
  $\tau \leftarrow$résoudre($\mathcal{P}$,τ_2)
  Si $\tau \neq \bot$
    afficher τ_2
    afficher τ_1
    afficher τ
    sortir
  Sinon
    enlever τ_1 et τ_2 de H
  Fin Si
Fin Pour

Procédure résoudre($\mathcal{P}, \tau_1$)
  Pour chaque solution τ_2 de $\mathcal{P}[\tau_1]$
    Si $\tau_2 \in H$
      renvoyer τ_2
    Sinon
      ajouter τ_2 à H
      $\tau \leftarrow$résoudre($\mathcal{P}$,τ_2)
      Si $\tau \neq \bot$
        afficher τ_2
        renvoyer τ
      Fin Si
      supprimer τ_2 de H
    Fin Si
  Fin Pour
  renvoyer $\bot$
Fin Procédure

FIGURE 8 – Résolution incrémentale d'un StCSP

sorte qu'une partie du système de transitions seulement soit calculée.

Un solveur de CSPs capable de générer l'ensemble des solutions d'un problème doit être capable de mémoriser la position courante dans l'espace de recherche de manière à pouvoir reprendre la recherche là où elle s'est arrêtée une fois que la solution courante a été affichée. Il est naturellement possible de faire plus que cela, notamment de résoudre un autre CSP paramétré par la solution trouvée.

L'algorithme présenté dans la figure 8 utilise une liste de hachage H pour mémoriser l'ensemble des états trouvés, afin de pouvoir détecter un cycle. La variable τ est utilisée pour repérer le cycle, qui est un état qui doit être déjà présent dans H. La valeur spéciale $\bot$ signifie que τ n'est pas assigné. Si τ est différent de $\bot$, cela signifie qu'une séquence $\tau_1\tau_2\ldots\tau_k$ satisfaisant toutes les contraintes a été trouvée, avec $\tau = \tau_i$ pour $0 \leq i \leq k$, ce qui représente un cycle dans le système de transitions.

L'algorithme travaille sur le CSP $\mathcal{P}(\mathcal{S}_R)$ associé au StCSP réduit, noté $\mathcal{P}$. Ce problème est composé d'un groupe de variables pour les valeurs à l'instant courant, et d'un groupe de variables pour les valeurs à l'instant suivant. L'algorithme commence par itérer l'ensemble des transitions. Pour chaque solution trouvée, il appelle la procédure `résoudre`. La notation $\mathcal{P}[\tau]$ désigne le problème $\mathcal{P}$ dans lequel le groupe de variables correspondant à l'instant courant est fixé à τ.

La solution est contenue dans la pile d'appel au moment où `résoudre` renvoit autre chose que $\bot$. L'algorithme produit un affichage de la solution, par l'intermédiaire des instructions `afficher`, qui doit être lu de bas en haut. L'état nécessaire pour déterminer le cycle est affiché en dernier.

Comme le nombre de solutions d'un CSP est fini, chaque boucle termine. La profondeur d'appel est également bornée par le nombre d'états du système de transitions. Ceci implique que l'algorithme finit toujours par s'arrêter. La complexité de cet algorithme est optimale en espace, car l'espace requis est linéaire en la taille de la solution trouvée. En revanche, il ne garantit pas de trouver une solution qui soit la plus courte. Il n'est pas non plus optimal en temps, car certains chemins du systèmes de transitions peuvent être explorés plusieurs fois, mais ceci peut être corrigé en utilisant une structure de données appropriée pour mémoriser les états qui mènent à des impasses. Enfin, une absence de cycle dans le systèmes de transitions implique que l'ensemble du système aura été parcouru. Le pire des cas est un système qui contient un maximum de transitions et aucun cycle, ce qui n'est pas fréquent, car dans la plupart des cas, un système de transitions contient soit peu de transitions, soit beaucoup de cycles.

8 Conclusion et perspectives

Nous avons présenté dans cet article un nouveau cadre pour la programmation par contraintes sur des variables flux. La figure 9 schématise les relations entres les différentes parties.

De nombreux formalismes existent concernant le raisonnement temporel. Les logiques temporelles sont très utilisées en vérification des modèles [4], mais les approches à basées contraintes sont virtuellement inexistantes. En particulier, bien que la transformation d'un k-StCSP en formule de logique temporel linéraire soit possible, elle rend impossible l'exploitation de la sémantique des contraintes.

Une littérature assez diverse existe en dehors du champ de la vérification de modèles. En particulier,

un formalisme a été développé concernant la satisfaction de formules de logique temporelle linéaire avec contraintes sur des structures particulières [3]. Bien que l'approche initiale dans [3] soit semblable, les structures, les types de contraintes et la manière de les résoudres sont très différents. En particulier, l'article s'appuie sur les méthodes habituellement utilisées pour la satisfaction de formules de logiques temporelles, qui ne sont pas basées sur la programmation par contraintes.

Les travaux de programmation par contraintes qui utilisent des contraintes temporelles sont très nombreux, mais sont dans la majorité des cas basés sur une approche avec horizon qui doit être repoussé indéfiniment, donnant lieu à des instances de plus en plus grandes du même problème[1].

Pour autant que nous puissions en juger, ce travail fait partie des premiers travaux qui concernent l'application de la programmation par contraintes à la modélisation de problèmes temporels sans horizon. Ce travail est fait en collaboration avec l'Université Chinoise de Hong Kong, et le formalisme décrit dans cet article est par conséquent fondamentalement similaire à celui décrit dans [7].

Le cadre des StCSPs étant nouveau, beaucoup de problèmes restent ouverts. Les possibilités concernant la modélisation restent à explorer, et les possibilités d'applications sont encore à l'étude, mais semblent nombreuses. Des sources d'inspiration sont les domaines de la planification, du contrôle, et de l'interaction, qui ne sont pas des domaines habituellement rencontrés en programmation par contraintes.

L'efficacité des algorithmes dans la pratique reste encore à mesurer. Des directions de recherches concernent le calcul d'une représentation compactes de l'ensemble des solutions basées sur les BDDs [8], mais le problème de l'explosion du nombre d'états est probablement beaucoup moins apparent en ce qui concerne la recherche d'une solution particulière.

Enfin, l'ensemble de toutes les solutions n'est pas toujours nécessaire. Le calcul d'un petit ensemble de solutions permettant de générer des solutions non périodiques serait suffisant pour des applications musicales, où la monotonie d'une solution unique, nécessairement périodique, serait d'un faible intérêt.

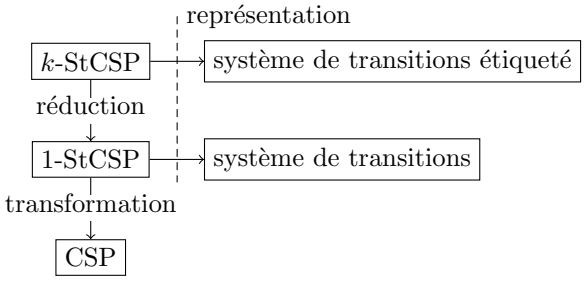


FIGURE 9 – Relations entre les formalismes

Références

[1] Krzysztof R. Apt and Sebastian Brand. Constraint-based qualitative simulation. *CoRR*, abs/cs/0504024, 2005.

[2] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling : Applying Constraint Programming to Scheduling Problems.* International Series in Operations Research and Management Science. Kluwer, 2001.

[3] Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint ltl. *Inf. Comput.*, 205(3) :380–415, 2007.

[4] E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B : Formal Models and Sematics (B)*, pages 995–1072. 1990.

[5] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice.* Elsevier, 2004.

[6] Mehdi Khiari, Patrice Boizumault, and Bruno Crémilleux. Constraint programming for mining n-ary patterns. In David Cohen, editor, *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 552–567. Springer, 2010.

[7] Arnaud Lallouet, Yat-Chiu Law, Jimmy Ho-Man Lee, and Charles F. K. Siu. Constraint programming on infinite data streams. In Toby Walsh, editor, *IJCAI*, pages 597–604. IJCAI/AAAI, 2011.

[8] K. L. Mcmillan. *Symbolic Model Checking : An Approach to the State Explosion Problem.* PhD thesis, Carnigie Mellon University, 1992.

[9] Cédric Pralet and Gérard Verfaillie. Using constraint networks on timelines to model and solve planning and scheduling problems. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric A. Hansen, editors, *ICAPS*, pages 272–279. AAAI, 2008.

Exploitation de la décomposition arborescente pour guider la recherche VNS

Mathieu Fontaine **Samir Loudni** **Patrice Boizumault**

Université de Caen Basse-Normandie, UMR 6072 GREYC, F-14032 Caen, France
CNRS, UMR 6072 GREYC, F-14032 Caen, France
{prénom.nom}@unicaen.fr

Résumé

La décomposition arborescente vise à découper un problème en *clusters* formant un graphe acyclique. Il existe des travaux exploitant la décomposition arborescente dans les méthodes complètes. Dans ce papier, nous montrons comment la decomposition arborescente peut être utilisée afin de guider l'exploration de méthodes de recherche locale à voisinages larges comme VNS. Nous présentons la *tightness dependent tree decomposition* (TDTD) qui permet de prendre en compte à la fois la structure du problème et la dureté des contraintes. Les expérimentations menées sur des instance aléatoires (GRAPH) et des instances réelles (CELAR et SPOT5) montrent la pertinence et l'efficacité de notre approche. *Cet article a été publié dans les actes d'ICTAI'2011 [5].*

Abstract

Tree decomposition introduced by Robertson and Seymour aims to decompose a problem into *clusters* constituting an acyclic graph. There are works exploiting tree decomposition for complete search methods. In this paper, we show how tree decomposition can be used to efficiently guide the exploration of local search methods that use large neighborhoods like VNS. We introduce tightness dependent tree decomposition which allows to take advantage of both the structure of the problem and the constraints tightness. Experiments performed on random instances (GRAPH) and real life instances (CELAR and SPOT5) show the appropriateness and the efficiency of our approach. *This paper has been published in the proceedings of ICTAI'2011 [5].*

1 Introduction

Un grand nombre de problèmes réels, tels que les problèmes d'affectation de fréquence [3] ou la planification d'un satellite d'observation de la terre [2], sont de très grande taille et exhibent un graphe de contraintes trés structuré. Exploiter ces propriétés structurelles peut aider à traiter ces problèmes.

La *décomposition arborescente*, proposée par Robertson et Seymour [17], vise à découper un problème en sous-problèmes (les *clusters*) constituant un graphe acyclique. Chaque *cluster* correspond à un sous ensemble de variables fortement connexes. Chaque sous-problème étant plus petit que le problème original, il est plus facile à résoudre. Un certain nombre de travaux exploitent la décomposition arborescente dans des méthodes de recherche complète, comme par exemple RDS-BTD [18], AND/OR graph search [12] et Decomposition Bounding (DB) [9].

Dans les méthodes de recherche locale qui utilisent des voisinages étendus tels que *Large Neighborhood Search* (LNS) [19], *Propagation guided large neighborhood search* (PGLNS) [16] ou encore *Variable Neighborhood Search* (VNS) [13], la définition de structures de voisinage pertinentes est une étape cruciale, car elle permet d'intensifier et de diversifier la recherche dans différentes régions de l'espace de recherche. À notre connaissance, il n'existe aucun travail exploitant la décomposition arborescente dans de telles méthodes.

Le cadre des *problèmes de satisfaction de contraintes pondérées* ($WCSP$ [10]) est un formalisme générique utilisé pour la modélisation et la résolution de problèmes d'optimisation sous contraintes. Il permet de traiter les problèmes sur-contraints et de modéliser les préférences entre solutions. Les $WCSP$ peuvent être résolus par des méthodes de recherche locale, hybride ou arborescente.

Dans ce papier, nous montrons comment la décomposition arborescente peut être utilisée afin de guider l'exploration de VNS. De plus, nous proposons la *tightness dependent tree decomposition* (TDTD) qui permet de prendre en compte à la fois la structure du

problème et la dureté des contraintes. Les expérimentations menées sur des instances aléatoires (GRAPH) et des instances réelles (CELAR et SPOT5) montrent que l'exploitation de la décomposition arborescente procure des performances bien meilleures comparé à VNS/LDS+CP [11] ou à ID-Walk [15]. À notre connaissance, nos travaux constituent la première tentative visant à utiliser la décomposition arborescente afin de guider une recherche locale de type VNS.

La section 2 présente le contexte de nos travaux. La section 3 montre comment exploiter la décomposition arborescente dans VNS. La TDTD est présentée dans la section 4. Les sections 5 et 6 présentent les résultats expérimentaux. Enfin, nous concluons et présentons des perspectives.

2 Définitions et notations

2.1 Problème de satisfaction de containtes pondérées (WCSP)

Un $WCSP$ est défini par un quadruplet $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{S}(k_\top))$, où $\mathcal{X}=\{x_1, ..., x_n\}$ est un ensemble de n *variables*, $\mathcal{D}=\{d_1, ..., d_n\}$ sont leurs *domaines finis respectifs*, $\mathcal{C}$ est un ensemble de *contraintes* sur $\mathcal{X}$, et $\mathcal{S}(k_\top)$ est une *structure de valuation*. $\mathcal{S}(k_\top)$ est un triplet ([[0,1,...,$k_\top$], $\oplus$, $\geq$) où : $k_\top$ est un entier naturel dans $[1, .., \infty]$, $\oplus$ est défini par : $a \oplus b = min(k_\top, a+b)$ et $\geq$ est l'opérateur d'ordre standard sur les entiers naturels. Chaque contrainte $c \in \mathcal{C}$ est défini par i) son *scope* $\mathcal{X}_c \subseteq \mathcal{X}$ (ensemble de variables sur lesquelles porte c), ii) sa *relation* $R_c = \prod_{x_i \in \mathcal{X}_c} d_i$ qui définit un ensemble d'affectations (ou tuples) de $\mathcal{X}_c$ et iii) sa *fonction de coût* $f_c : R_c \mapsto [0, k_\top]$. L'affectation de x_i à une valeur $a \in d_i$ est notée $(x_i{=}a)$. Une affectation *complète* $\mathcal{A}{=}(a_1, ..., a_n)$ est une affectation de toutes les variables ; dans le cas contraire, on l'appelle affectation *partielle*. $\mathcal{A}_{\downarrow \mathcal{X}_c}$ désigne une affectation partielle obtenue en projetant l'affectation complète $\mathcal{A}$ sur les variables de la contrainte c. Pour une affectation complète $\mathcal{A}$, si $f_c(\mathcal{A}_{\downarrow \mathcal{X}_c}) = 0$, c est dite satisfaite, sinon elle est violée. Le coût d'une affectation complète $\mathcal{A}{=}(a_1, ..., a_n)$ est défini par $f(A) = \sum_{c \in \mathcal{C}} f_c(\mathcal{A}_{\downarrow \mathcal{X}_c})$. L'objectif est de trouver une affectation complète de coût minimal : $min_{\mathcal{A} \in d_1 \times d_2 \times ... \times d_n} f(\mathcal{A})$.

2.2 Décomposition arborescente

Le graphe de contraintes d'un $WCSP$ est un graphe $G{=}(\mathcal{X}, E)$ composé d'un sommet par variable et d'une arête (u, v) pour chaque contrainte $c \in \mathcal{C}$ telle que $u, v \in \mathcal{X}_c$.

Definition 1 *Une décomposition arborescente [17] de $G{=}(\mathcal{X}, E)$ est un couple (C_T, T) où :*

- *$T = (I, A)$ est un arbre avec pour ensemble de nœuds I et pour ensemble d'arêtes A,*
- *$C_T = \{C_i \mid i \in I\}$ est une famille de sous-ensembles de $\mathcal{X}$ (appelés clusters) telle que :*
  - *$\cup_{i \in I} C_i = \mathcal{X}$,*
  - *$\forall (u, v) \in E$, $\exists C_i \in C_T$ t.q. $u, v \in C_i$,*
  - *$\forall i, j, k \in I$, si j est sur le chemin de i à k dans T, alors $C_i \cap C_k \subseteq C_j$.*

Definition 2 *L'intersection entre deux clusters est appelée séparateur, et est notée $sep(C_i, C_j)$. Deux clusters sont adjacents s'ils partagent au moins une variable. Le voisinage d'un cluster C_i dans T est $\mathcal{N}(C_i){=}\{C_j \mid j \in I, sep(C_i, C_j) \neq \emptyset\}$. De la même façon, pour un ensemble C_s de clusters dans T, $\mathcal{N}(C_s){=}\cup_{C_i \in C_s} \mathcal{N}(C_i)$. Les variables appartenant à un et un seul cluster sont appelées variables propres.*

La largeur d'une décomposition arborescente $T{=}(I, A)$ est définie par $w(T){=}\max_{i \in I}(|C_i|{-}1)$. La largeur de décomposition $tw(G)$ d'un graphe G est la plus petite largeur de toutes les décompositions arborescentes possibles de G. Calculer la largeur de décomposition d'un graphe est un problème NP-complet [1]. Cependant, des heuristiques reposant sur la notion de *triangulation* de graphe [1] permettent le calcul de décompositions approchées. Elles fournissent un majorant de la largeur de décomposition.

Dans cet article, nous utilisons l'heuristique *maximum cardinality search* (MCS) [20] qui constitue un bon compromis entre la largeur de la décomposition obtenue et le temps nécessaire à son calcul [8].

La Fig. 1 illustre les trois étapes nécessaires au calcul d'une décomposition d'un graphe G (a). Tout d'abord, le graphe G est triangulé par ajout d'arêtes (b). Ensuite, on calcule les cliques maximales afin de constituer le graphe de *clusters* (c). Enfin, on obtient la décomposition arborescente de G par calcul de l'arbre de jointure (d).

2.3 VNS/LDS+CP

VNS/LDS+CP [11] est une méthode de recherche locale qui exploite le principe de VNDS (*Variable Neighborhood Decomposition Search*) [6]. Les voisinages sont obtenus en désaffectant une partie de la solution courante selon une heuristique de choix de voisinage. Ensuite, l'exploration de l'espace de recherche est réalisée par une recherche arborescente partielle LDS (*Limited discrepancy search*, [7]) aidée par une propagation des contraintes (CP) basée sur un calcul de minorants.

L'algorithme 1 présente le pseudo-code de VNS/LDS+CP. Soit $\mathcal{X}$ l'ensemble des variables et

1. Un graphe est cordal ou triangulé si et seulement si tout ses cycles de taille supérieure à quatre ont une corde (une arête connectant deux sommets non-adjacents du cycle).

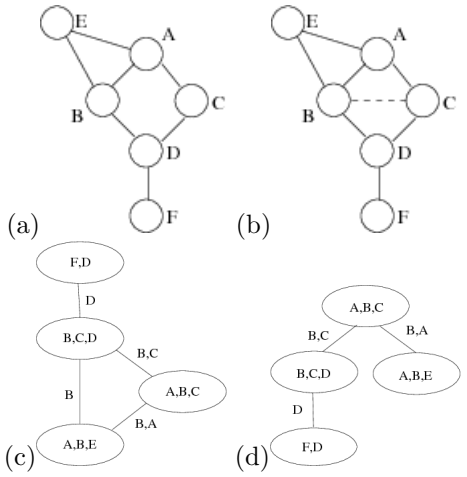


FIGURE 1 – (a) Graphe initial G. (b) Exemple de triangulation de G. (c) Cliques maximales du graphe triangulé (*graphe de clusters*). (d) Décomposition arborescente de G de largeur 2.

Algorithm 1: VNS/LDS+CP

```
   fonction VNS/LDS+CP(X, C, k_init, k_max, δ_max) ;
   begin
1  |  S ← genInitSol() ;
2  |  k ← k_init ;
3  |  while (k < k_max) ∧ (notTimeOut) do
4  |  |  X_un ← Hneighborhood(X, N_k, S) ;
5  |  |  A ← S\{(x_i = a) | x_i ∈ X_un} ;
6  |  |  S' ← LDS+CP(A, X_un, δ_max, f(S), S) ;
7  |  |  if f(S') < f(S) then
8  |  |  |  S ← S' ;
9  |  |  |  k ← k_init ;
10 |  |  else k ← k + 1 ;
11 |  return S ;
   end
```

N_k la structure de voisinage de dimension k ; N_k correspond à l'ensemble des sous-ensembles de k variables de $\mathcal{X}$. VNS/LDS+CP part d'une solution initiale S aléatoire (ligne 1). Un sous-ensemble de k variables est sélectionné dans $\mathcal{X}$ par l'heuristique de choix de voisinage Hneighborhood (ligne 4). Une affectation partielle $\mathcal{A}$ est générée à partir de la solution courante S en désaffectant les k variables sélectionnées ; Les $(n-k)$ variables restantes gardent leur valeur dans S (ligne 5). Ensuite, la solution est reconstruite (ligne 6) par une recherche arborescente partielle (LDS) aidée par une propagation des contraintes (CP) basée sur le calcul de minorants. Si cette recherche trouve une solution de meilleure qualité S' dans le voisinage de S (ligne 7), S' devient la solution courante et k est réinitialisé à k_{init} (lignes 8-9). Sinon, on cherche de nouvelles améliorations dans N_{k+1} (structure de voisinage de taille $k+1$)(ligne 10)). La recherche s'arrête quand elle a atteint k_{max} la taille maximale de voisinage ou la limite de temps (ligne 3).

2.4 Heuristique de choix de voisinage

L'heuristique de choix de voisinage, utilisée pour sélectionner les variables à désaffecter, guide VNDS dans l'exploration de l'espace de recherche afin de trouver des solutions de meilleure qualité. ConflictVar est une des heuristiques les plus simples : pour une dimension de voisinage k, ConflictVar sélectionne aléatoirement k variables à désaffecter parmi celles en conflit. Une telle heuristique, basée sur des choix aléatoires, permet de diversifier la recherche et de sortir rapidement des optima locaux.

3 Intensification/Diversification exploitant la décomposition arborescente

Dans cette section, nous présentons la première contribution de notre article : DGVNS (Decomposition Guided VNS) qui utilise le graphe de *clusters* afin de construire des structures de voisinage permettant une meilleure intensification et une meilleure diversification que VNS/LDS+CP. Tout comme VNS/LDS+CP, DGVNS utilise une recherche arborescente partielle (LDS+CP) pour la phase de reconstruction, mais les voisinages sont gérés différemment afin de tirer parti du graphe de *clusters*.

Au lieu d'utiliser les structures de voisinage N_k de VNS/LDS+CP (voir section 2.3), DGVNS utilise des structures de voisinage $N_{k,i}$, où k est la dimension du voisinage et C_i désigne le *cluster* dans lequel les variables à désaffecter vont être sélectionnées. L'algorithme 2 décrit le pseudo-code de DGVNS.

Nous favorisons les mouvements dans des régions fortement liées. Le *cluster* est une structure qui permet d'identifer ces régions, du fait de sa taille (plus petite que le problème original), et du lien fort entre les variables qu'il contient. Nous sélectionnons tout d'abord les k variables à désaffecter dans un même *cluster* C_i. Si $(k > |C_i|)$, alors l'ensemble des variables potentielles C_s est complété en ajoutant les variables des *clusters* C_j adjacents à C_i afin de prendre en compte la topologie du graphe de *clusters*. Ce traitement est accompli par la fonction CompleteCluster(C_i) (ligne 7). La structure de voisinage $N_{k,i}$ est constituée de l'ensemble des sous-ensembles de k variables de C_s (ligne 8).

le but de la diversification est de traiter un grand nombre de régions différentes, afin d'assurer que l'espace de recherche est correctement exploré, et d'essayer de localiser la région contenant l'optimum global.

Algorithm 2: DGVNS

```
   fonction DGVNS(X, C, k_init, k_max, δ_max);
   begin
1      Soit G le graphe de contraintes de P ;
2      Soit (C_T, T) une décomposition arborescente de G ;
       Soit C_T = {C_1, C_2, ..., C_p} ;
3      S ← genInitSol() ;
4      k ← k_init ;
5      i ← 1 ;
6      while (k < k_max) ∧ (notTimeOut) do
7          C_s ← CompleteCluster(C_i) ;
8          X_un ← Hneighborhood(C_s, N_{k,i}, S) ;
9          A ← S\{(x_i = a) | x_i ∈ X_un} ;
10         S' ← LDS+CP(A, X_un, δ_max, f(S), S) ;
11         if f(S') < f(S) then
12             S ← S' ;
13             k ← k_init ;
14             i ← succ(i) ;
15         else k ← k + 1 ; i ← succ(i) ;
16     return S ;
   end
```

Afin d'améliorer la diversification, nous considérons successivement tous les C_i.

Soit p le nombre total de *clusters*, *succ* est la fonction qui associe à un *cluster* i son successeur [2], et $N_{k,i}$ la structure de voisinage courante : si LDS+CP trouve une solution de meilleure qualité S' dans le voisinage de S (ligne 11), alors S' devient la solution courante (ligne 12), k est réinitialisé à k_{init} (ligne 13), et le *cluster* suivant est considéré (ligne 14). Sinon, on cherche des améliorations dans $N_{(k+1),succ(i)}$ (structure de voisinage de taille $(k+1)$) (ligne 15). La recherche s'arrête quand elle atteint k_{max} la taille maximale de voisinage ou la limite de temps (ligne 6).

Tout d'abord, la diversification réalisée par le déplacement du *cluster* C_i vers le *cluster* $C_{succ(i)}$ est nécessaire. Les expérimentations que nous avons menées montrent que rester dans un même *cluster* donne de moins bons résultats : sélectionner un nouveau *cluster* permet d'améliorer la qualité de la solution en visitant de nouvelles parties de l'espace de recherche.

Puis, quand un minimum local est atteint, passer de k à $(k{+}1)$ permet de renforcer la diversification par l'élargissement de la taille du voisinage.

Le but de l'intensification est de trouver la meilleure solution dans une petite région de l'espace de recherche. Ceci est réalisé par la reconstruction de la solution partielle avec LDS+CP et par la réduction de la taille de voisinage à k_{init} à chaque amélioration.

2. si $i < p$ alors $succ(i) = i + 1$ sinon $succ(p) = 1$.

4 Tightness Dependent Tree Decomposition

Les problèmes d'optimisation contiennent souvent des sous-problèmes plus durs à résoudre que les autres en raison de la *dureté* de leur contraintes. Ces sous-problèmes restent cachés dans la décomposition. Pour identifier ces sous-parties difficiles, on utilise *la dureté de la contrainte*. Soit c une contrainte et R_c sa relation. La dureté de la contrainte c est définie par [3] :

$$t(c) = \frac{|\{t \,|\, f_c(t) > 0, t \in R_c\}|}{|\{t \in R_c\}|} \quad (1)$$

Notre approche procède en deux temps :

1. Toutes les contraintes de dureté inférieure à un seuil (noté λ) sont retirées du graphe initial.
2. Le graphe ainsi obtenu est décomposé.

La fig. 2 montre l'influence du seuil λ sur la décomposition de l'instance Scen06. La col. 1 donne les différentes valeurs de λ. La col. 2 indique le pourcentage de contraintes retirées. La col. 3 présente le nombre total de *clusters*. Les col. 4-12 reportent respectivement, pour chacun des trois paramètres (*taille des clusters*, *degré du cluster* et *taille des séparateurs*), leurs valeurs minimale, moyenne et maximale. Enfin la dernière colonne indique le nombre de séparateurs. La décomposition arborescente sur le graphe initial correspond au seuil λ=0.

Pour $0,2 \leq \lambda \leq 0,5$, la TDTD donne le plus souvent un grand nombre de *clusters* de petite taille, tout en conservant les contraintes les plus importantes du problème initial. De plus, les *clusters* ont (en moyenne) un plus haut degré (une connexité plus grande) que ceux obtenus avec $\lambda = 0$. Ceci montre l'intérêt et l'importance de la TDTD. Cependant, pour des valeurs élevées de λ ($\lambda \geq 0,6$), elle n'est pas pertinente car trop de contraintes sont retirées, conduisant à des *clusters* isolés de taille négligeable. Les résultats obtenus sur les autres instances du CELAR ne sont pas reportés dans cet article car ils sont très similaires. Sur les instances SPOT5, la TDTD est pertinente pour $0,1 \leq \lambda \leq 0,3$ (voir sect. 6.3 pour plus de détails).

5 Jeux de test

Les expérimentations ont été menées sur les instances de trois problèmes différents :

3. Notre définition courante de la dureté d'une contrainte ne prend pas en compte les fonctions de coût (voir les perspectives, section 7).

λ	% dropped	$\lvert C_T \rvert$	Taille des *clusters*			Degré des *clusters*			Taille des séparateurs			
	c		min.	moy.	max.	min.	moy.	max.	min.	moy.	max.	nb
0	0	55	2	4,9	12	1	9,6	25	1	1,77	8	266
0,1	1	59	2	5,1	11	1	10,27	26	1	1,9	10	303
0,2	3	61	2	4,9	11	1	10,06	26	1	1,89	10	307
0,3	5	61	2	4,91	11	1	10,06	26	1	1,8	10	307
0,4	8	60	1	4,75	11	0	7,8	21	1	1,9	10	234
0,5	14	56	1	4,3	11	0	5,57	14	1	1,75	9	156
0,6	28	65	1	3,93	10	0	5,29	14	1	1,93	9	172
0,7	53	74	1	2,98	10	0	4,10	10	1	1,61	6	152
0,8	54	72	1	2,97	10	0	3,63	9	1	1,64	6	131
0,9	76	80	1	1,85	9	0	1,6	7	1	1,18	5	64
1	100	100	1	1	1	0	0	0	0	0	0	0

FIGURE 2 – TDTD de l'instance Scen06.

Instance	Méthode	Succ.	Temps	Moy.
Scen06	DGVNS	50/50	112	3389
$n = 100$	SDGVNS-0, 2	**50/50**	**58**	**3389**
$e = 1222$	VNS/LDS+CP	15/50	83	3399
$S^* = 3389$	ID-Walk	NA	840	3447 (3389)
Scen07	DGVNS	40/50	317	345614
$n = 200$	SDGVNS-0, 4	**49/50**	**221**	**343600**
$e = 2665$	VNS/LDS+CP	1/50	461	355982
$S^* = 343592$	ID-Walk	NA	360	373334 (343998)
Scen08	DGVNS	3/50	1811	275
$n = 458$	SDGNVS-0, 5	**9/50**	**442**	**272**
$e = 5286$	VNS/LDS+CP	0/50	-	394 (357)
$S^* = 262$	ID-Walk	NA	3000	291 (267)

FIGURE 3 – Comparaison entre DGVNS, VNS/LDS+CP et ID-Walk sur les instances RLFAP.

Instances RLFAP : Le CELAR (Centre d'électronique de l'Armement) a mis à disposition un ensemble d'instances du problème d'affectation de fréquences radio (RLFAP) [3]. L'objectif est d'assigner un nombre limité de fréquences à un ensemble de liens radios entre des paires de sites, afin de minimiser les interférences dues à la réutilisation des fréquences. Nous reportons les résultats sur les instances les plus difficiles : Scen06, Scen07 et Scen08.

Instances GRAPH : Le générateur GRAPH (Generating Radio link frequency Assignment Problems Heuristically) a été développé par le projet CALMA [21] afin de proposer des instances aléatoires ayant une structure proche des instances RLFAP.

Instances SPOT5 : La planification quotidienne d'un satellite d'observation de la terre, SPOT5, consiste à sélectionner les prises de vue à effectuer dans la journée en prenant en compte les limites matérielles du satellite tout en maximisant l'importance des photographies sélectionnées [2]. Nous avons réalisé des expérimentations sur les instances SPOT5 afin de confirmer les conclusions tirées à partir des instances RLFAP. Nous reportons les résultats sur six instances sans contraintes dures de capacité.

6 Expérimentations

6.1 Protocole expérimental

Chaque méthode a été appliquée sur chaque instance, avec une *discrepancy* de 3 pour LDS, ce qui correspond à la meilleure valeur trouvée sur les instances RLFAP [11]. Les valeurs de k_{min} et k_{max} ont été respectivement fixées à 4 et n (le nombre total de variables), et le temps de calcul maximal à 3600 secondes. Un ensemble de 50 essais par instance a été réalisé sur un AMD opteron 2,1 GHz et 256 Go de RAM. Toutes les méthodes ont été implantées en C++ en utilisant la librairie toulbar2 [4].Pour ID-Walk, Nous avons utilisé la librairie INCOP [14]. Pour chaque instance et chaque méthode, nous reportons le nombre d'essais réussis (c'est-à-dire ayant atteint l'optimum), le temps de calcul moyen pour atteindre l'optimum ainsi que le coût moyen des solutions trouvées sur les 50 essais et, si nécessaire, le meilleur coût obtenu (noté entre parenthèse) dans le cas où l'optimum n'a pas été atteint. Nous présentons aussi les *profils de performance moyens* sur les 50 essais montrant l'évolution de la qualité des solutions en fonction du temps.

Dans un premier temps, nous comparons DGVNS avec VNS/LDS+CP et ID-Walk [15], une des méthodes de recherche locale les plus efficaces sur les instances RLFAP (section 6.2). Dans un second temps, nous comparons DGVNS avec la version utilisant la TDTD (SDGVNS-λ). Nous présentons les résultats pour différentes valeurs du seuil λ (section 6.3). Ensuite, nous étudions l'influence de la largeur de décomposition sur notre approche (section 6.4). Il est à noter que la comparaison des temps de calcul entre notre méthode et des méthodes de recherche complète serait peu pertinente. En effet, ces dernières vont à la fois trouver la ou les solutions optimales et prouver leur optimalité. Ces opérations peuvent prendre quelques jours sur ces instances [18].

4. http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP

Instance	Méthode	Succ.	Temps	Moy.
408	DGVNS	49/50	117	6228
$n = 200$	SDGVNS-0, 1	**50/50**	**72**	**6228**
$e = 2232$	VNS/LDS+CP	26/50	149	6228
$S^* = 6228$	ID-Walk	**50/50**	**3**	**6228**
412	DGVNS	36/50	84	32381
$n = 300$	SDGVNS-0, 1	**38/50**	**71**	**32381**
$e = 4348$	VNS/LDS+CP	32/50	130	32381
$S^* = 32381$	ID-Walk	10/50	2102	3238
414	DGVNS	38/50	554	38478
$n = 364$	SDGNVS-0, 1	**42/50**	**430**	**38478**
$e = 10108$	VNS/LDS+CP	12/50	434	38481
$S^* = 38478$	ID-Walk	0/50	-	38481
505	DGVNS	**50/50**	**63**	**21253**
$n = 240$	SDGVNS-0, 1	48/50	92	21253
$e = 2242$	VNS/LDS+CP	41/50	143	21253
$S^* = 21253$	ID-Walk	**50/50**	**358**	**21253**
507	DGVNS	33/50	71	27390
$n = 311$	SDGVNS-0, 1	**45/50**	**62**	**27390**
$e = 5732$	VNS/LDS+CP	11/50	232	27391
$S^* = 27390$	ID-Walk	7/50	1862	27391
509	DGVNS	**40/50**	**265**	**36446**
$n = 348$	SDGVNS-0, 2	**39/50**	**313**	**36446**
$e = 8624$	VNS/LDS+CP	12/50	598	36448
$S^* = 36446$	ID-Walk	0/50	-	36450

FIGURE 4 – Comparaison entre DGVNS, VNS/LDS+CP et ID-Walk sur les instances SPOT5.

Nos expérimentations montrent :

1. l'efficacité de notre approche comparée à VNS/LDS+CP et ID-Walk sur des instances structurées telles que RLFAP et SPOT5 ;
2. l'utilité de la TDTD pour créer des structures de voisinage pertinentes.

6.2 Apport de la décomposition arborescente

Sur les instances RLFAP, DGVNS surclasse VNS/LDS+CP (fig. 3). DGVNS atteint l'optimum sur 100% des essais pour la Scen06, 80% pour la Scen07 et 6% pour la scen08. VNS/LDS+CP n'obtient l'optimum que dans de rares cas sur les deux premières instances et ne l'obtient pas pour la Scen08 : la meilleure solution trouvée a un coût de 357. Cette tendance se confirme sur les instances SPOT5 (fig. 4) pour lesquelles DGVNS devance VNS/LDS+CP, en terme de taux de réussite (avec un gain de 40% en moyenne) et en terme de temps de calcul. En effet, DGVNS atteint l'optimum sur les 50 essais sur deux instances (408 et 505). Pour les autres instances (sauf la 507), le taux de réussite est au moins de 70%.

De plus, DGVNS dépasse nettement les performances d'ID-Walk [5], notamment sur les deux instances les plus

Instance	λ	Succ.	Temps	Moy.
Scen06	0, 2	**50/50**	**58**	**3389**
	0, 3	**50/50**	**61**	**3389**
	0, 4	50/50	110	3389
	0, 5	49/50	122	3389
Scen07	0, 2	45/50	271	344603
	0, 3	**47/50**	**229**	**344198**
	0, 4	**49/50**	**221**	**343600**
	0, 5	45/50	244	344603
Scen08	0, 2	**7/50**	**327**	273
	0, 3	5/50	323	273
	0, 4	6/50	344	274
	0, 5	**9/50**	**442**	**272**
408	0, 1	**50/50**	**72**	**6228**
	0, 2	**44/50**	**90**	**6228**
	0, 3	10/50	105	6229
412	0, 1	**38/50**	**71**	**32381**
	0, 2	32/50	135	32384
	0, 3	8/50	1337	32384
414	0, 1	**42/50**	**430**	**38478**
	0, 2	**38/50**	**471**	**38478**
	0, 3	4/50	704	38480
505	0, 1	**48/50**	**92**	**21253**
	0, 2	**18/50**	**148**	**21253**
	0, 3	4/50	319	21256
507	0, 1	**45/50**	**62**	**27390**
	0, 2	**28/50**	**134**	**27390**
	0, 3	3/50	418	27391
509	0, 1	**36/50**	**286**	**36446**
	0, 2	**39/50**	**313**	**36446**
	0, 3	2/50	583	36448

FIGURE 5 – Influence du seuil λ sur la TDTD

difficiles, Scen07 et Scen08 (Fig. 3). Pour la Scen07 (resp. Scen08), DGVNS obtient des solutions ayant une *déviation moyenne* (pourcentage de déviation par rapport à l'optimum) de 0, 55% (resp. 5%), tandis qu'ID-Walk obtient des solutions avec une déviation moyenne de 8% (resp. 11%).

Sur les instances SPOT5 (sauf la 408), DGVNS devance ID-Walk (Fig. 4), particulièrement sur les instances 414 et 509, où ID-Walk ne trouve jamais l'optimum. Pour ces deux instances, les meilleures solutions trouvées ont un coût respectif de 38479 et 36447.

Les fig. 10 et 11 comparent les profiles de performance de DGVNS et VNS/LDS+CP. DGVNS surpasse VNS/LDS+CP sur les problème RLFAP et SPOT5 (sauf pour l'instance 505). D'un point de vue *anytime*, on observe deux phénomènes importants. Premièrement, la courbe associée au comportement de DGVNS présente une plus forte pente. De plus, la décélération de la courbe associée au comportement de VNS/LDS+CP est beaucoup plus forte que celle de DGVNS. Cela confirme la pertinence de l'utilisation de la décomposition arborescente pour orienter l'exploration dans VNS.

5. Les résultats pour les instances RLFAP sont tirés de [15]. Le nombre de réussites et les temps de calcul ne sont pas disponibles (NA Fig. 3), Nous ne donnons donc que le temps par essai, le coût moyen obtenu sur 10 essais et le meilleur coût (entre parenthèses). Pour les autres instances (SPOT5 et GRAPH), les résultats ont été obtenus avec le protocole expérimental décrit précédemment.

6.3 Apport de la *tightness dependent tree decomposition* (TDTD)

Les fig. 3 et 4 comparent également SDGVNS et DGVNS. On reporte ici les valeurs associées aux seuils λ ayant donné les meilleurs résultats.

Sur les instances RLFAP, l'apport de la TDTD est très important, particulièrement sur les instances Scen07 et Scen08, pour lesquelles on observe de fortes améliorations en comparaison des résultats de DGVNS. Pour la Scen07, SDGVNS améliore de 18% le taux de réussite (de 80% à 98%) et réduit la déviation moyenne à l'optimum (de $0,55\%$ à $0,002\%$). Pour la Scen08, le taux de réussite est amélioré de 12% (de 6% à 18%), la déviation moyenne descend de 5% à $3,80\%$ et SDGVNS est 4 fois plus rapide. Pour la Scen06, SDGVNS divise par deux le temps de calcul. Une fois encore, cette tendance se confirme sur les instances SPOT5, sur lesquelles SDGVNS se montre très efficace. Sur les grandes instances (412, 414 et 507), SDGVNS améliore le nombre moyen de réussites ainsi que le temps de calcul de 12% et 25%. Sur les autres instances, les deux méthodes montrent des performances similaires. Cependant, SDGVNS est au moins $1,5$ fois plus rapide que DGVNS. Du point de vue des profils de performance (fig. 10 et 11), on observe que SDGVNS a systématiquement un meilleur comportement *anytime* que DGVNS.

La fig. 5 montre l'impact du seuil λ sur la TDTD. La meilleure valeur pour le seuil dépend de l'instance considérée. Les expérimentations montrent que pour les instances RLFAP, une valeur de λ comprise entre $0,2$ et $0,5$ donne les meilleurs résultats. Pour les instances SPOT5, on obtient les meilleures performances avec un seuil fixé entre $0,1$ et $0,3$. En effet, ces instances sont très contraintes car seulement un petit nombre de prises de vue peuvent être sélectionnées dans une solution optimale. Un seuil de $0,3$ (resp. $0,1$) donne des résultats plus homogènes sur les instances RLFAP (resp. SPOT5). Enfin, pour les instances RLFAP (resp. SPOT5), un seuil fixé au-delà de $0,6$ (resp. $0,4$) ne donne pas de bons résultats car trop de contraintes sont retirées. Nous ne reportons donc pas les résultats correspondants dans cet article.

6.4 Impact de la largeur de décomposition

Comme nous l'avons précisé dans l'introduction, les problèmes réels présentent souvent une structure particulière, notamment des sous-structures peu connectées entre elles (fig. 6). Ces structures reflètent des propriétés topologiques induites par la décomposition arborescente et se caractérisent par trois mesures : la largeur de la décomposition, la taille des séparateurs, et le degré des *clusters*. La largeur de décomposition donne une bonne indication sur la taille des

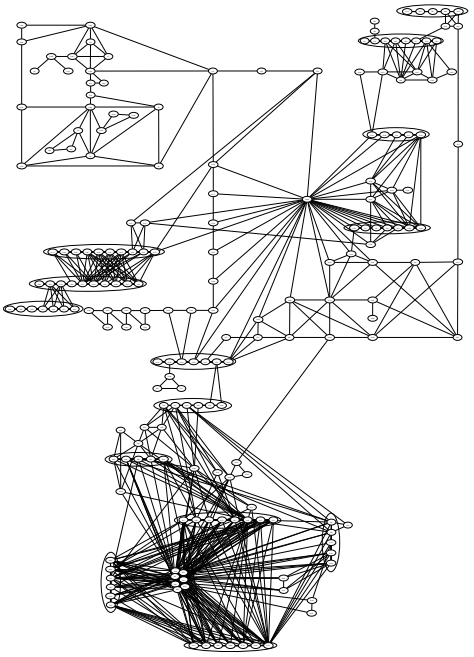


FIGURE 6 – Le graphe de contrainte de l'instance Scen07. Les variables contenues dans une ellipse forment une clique.

Instance	n	w^-	mc	$\frac{w^-}{n}$	$\frac{mc}{n}$
Scen06	100	11	10	0,11	0,10
Scen07	200	18	10	0,09	0,05
Scen08	458	18	10	0,03	0,02
Graph05	100	28	8	0,28	0,08
Graph06	200	54	8	0,27	0,04
Graph11	340	102	7	0,3	0,02
Graph13	458	136	6	0,29	0,01

FIGURE 7 – Plus petite largeur de décomposition, taille et ratios pour les instances RLFAP et GRAPH.

sous-problèmes, tandis que les deux autres mesures permettent d'évaluer la connexité entre *clusters*. En effet, plus ces valeurs sont basses, moins les *clusters* sont connectés. Comme il est indiqué dans [4], le calcul d'une décomposition arborescente optimisant ces trois mesures est une question ouverte. Nous avons sélectionné l'heuristique MCS [20] car celle-ci présente un bon compromis entre la qualité de la décomposition et le temps nécessaire pour son calcul [8].

Pour chaque instance *inst* RLFAP, SPOT5 et GRAPH, nous avons calculé toutes les décompositions fournies par MCS. Pour une instance *inst*, soit $w^-(inst)$ la plus petite largeur de décomposition obtenue par MCS. Le tableau 7 indique pour chaque instance : la valeur de w^-, la taille de la plus grande clique dans le graphe

de contraintes (mc), la taille du problème (n), et les ratios ($\frac{w^-}{n}$) et ($\frac{mc}{n}$) pour les instances RLFAP et GRAPH. Les résultats pour les instances SPOT5 sont très similaires à ceux des instances RLFAP, ils ne sont donc pas reportés ici. $(mc-1)$ est un minorant de la largeur de décomposition. Elle donne une bonne indication sur la qualité de la décomposition obtenue.

Sur les instances RLFAP (fig. 7), la plus petite largeur w^- augmente avec la taille du problème (n), tandis que le ratio ($\frac{w^-}{n}$) diminue. Ce qui signifie que pour les plus grandes instances, les *clusters* sont plus dispersés. Au contraire, pour les instances GRAPH, les valeurs de w^- ainsi que leurs ratios sont très élevés comparés aux instances RLFAP, particulièrement pour les instances Graph11 et Graph13. Ce tableau montre aussi de grands écarts (> 100) entre les minorants et les majorants de la largeur de décomposition sur les grandes instances GRAPH, alors que cet écart est au plus de 8 pour les instances RLFAP. Les mauvais résultats sur les instances Graph11 et Graph13 montrent l'impossibilité pour l'heuristique MCS d'exhiber précisément une structure pertinente pour ces problèmes. Cette différence notable entre les instances les plus grandes des problèmes RLFAP et GRAPH semble provenir de l'origine de ces deux jeux de test : les instances RLFAP sont issues de problèmes réels, tandis que les instances GRAPH sont générées aléatoirement.

Comme nous l'avons montré dans les sections 6.2 et 6.3, DGVNS et SDGVNS surpassent VNS/LDS+CP et ID-Walk sur les instances ayant de faibles valeurs de w^- (instances RLFAP et SPOT5). Ceci se confirme sur les instances Graph05 et Graph06 en terme de taux de réussite et de qualité de solution (fig. 8) mais aussi au niveau des profils de performance (fig. 12).

Cependant, sur les instances Graph11 et Graph13, VNS/LDS+CP montre de meilleures performances que DGVNS et SDGVNS (fig. 8). Cela s'explique par le fait que les *clusters* sont de très grande taille et fortement connexes. Ainsi, l'impact de notre méthode n'est pas aussi fort car les *clusters* se recouvrent fortement et sont donc très semblables. On peut néanmoins remarquer que la comparaison des profils de performance (fig. 12) montre que le comportement de SDGVNS est bien meilleur au niveau de l'évolution de la qualité de la solution au fil du temps que celui de VNS/LDS+CP. Des travaux futurs devront être menés afin de tirer des conclusions plus précises.

Enfin, la fig. 9 décrit l'impact du seuil λ sur la TDTD pour les instances GRAPH. Les meilleurs résultats sont obtenus avec des valeurs de λ élevées $(0, 7)$. En effet, les *clusters* étant relativement denses, un grand nombre de contraintes doivent être retirées pour obtenir une décomposition pertinente.

Instance	Méthode	Succ.	Temps	Moy.
Graph05	DGVNS	50/50	10	221
$n = 100$	SDGVNS-0, 7	**50/50**	**8**	**221**
$e = 1034$	VNS/LDS+CP	50/50	17	221
$s* = 221$	ID-Walk	0/50	-	10584 (2391)
Graph06	DGVNS	50/50	367	4123
$n = 200$	SDGVNS-0, 7	50/50	261	4123
$e = 1970$	VNS/LDS+CP	**50/50**	**218**	4123
$s* = 4123$	ID-Walk	0/50	-	18949 (13001)
Graph11	DGVNS	8/50	3046	4234
$n = 340$	SDGVNS-0, 5	12/50	2818	3268
$e = 3417$	VNS/LDS+CP	**44/50**	2403	3090
$s* = 3080$	ID-Walk	0/50	-	41604 (27894)
Graph13	DGVNS	0/50	-	22489 (18639)
$n = 458$	SDGVNS-0, 7	0/50	-	11449 (10268)
$e = 4915$	VNS/LDS+CP	**3/50**	3477	14522
$s* = 10110$	ID-Walk	0/50	-	58590 (47201)

FIGURE 8 – Comparaison de DGVNS, VNS/LDS+CP et ID-Walk sur les instances GRAPH.

Instance	λ	Succ.	Temps	Moy.
Graph05	0, 5	50 / 50	12	221
	0, 6	50 / 50	13	221
	0, 7	**50 / 50**	**8**	**221**
Graph06	0, 5	50 / 50	339	4123
	0, 6	49 / 50	259	4123
	0, 7	**50 / 50**	**261**	**4123**
Graph11	0, 5	**12 / 50**	**2818**	**3643**
	0, 6	**7 / 50**	**2446**	**3123**
	0, 7	5 / 50	1417	3223
Graph13	0, 5	0 / 50	-	24637 (18833)
	0, 6	0 / 50	-	19340 (14318)
	0, 7	**0 / 50**	**-**	**11449 (10268)**

FIGURE 9 – Impact du seuil λ sur la TDTD.

7 Conclusion

Nous avons montré que la *décomposition arborescente* permet de guider efficacement l'exploration de l'espace de recherche. De plus, nous avons proposé la *tightness dependent tree decomposition* qui permet de prendre en compte à la fois la structure du problème et la dureté des contraintes. Notre méthode d'intensification/diversification est générique et peut être appliqué à d'autre méthodes de recherche locale telle que LNS ou PGLNS. Enfin, nous avons montré expérimentalement que l'exploitation des décompositions arborescentes permet de dépasser les performances de VNS/LDS+CP et d'ID-Walk.

Nous travaillons actuellement sur trois pistes : tirer parti des séparateurs et des variables propres des *clusters* pour orienter la recherche, paralléliser l'exploration des *clusters*, et étudier l'apport de la méthode *min-fill* pour améliorer la qualité de la décomposition. Il serait intéressant d'étudier la pertinence de la prise en compte des coûts, comme proposé dans [9], pour guider la décomposition.

Ce travail a été soutenu par l'Agence Nationale de la Recherche, référence ANR-10-BLA-0214.

Références

[1] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM. J. on Algebraic and Discrete Methods*, 8 :277–284, 1987.

[2] E. Bensana, M. Lemaître, and G. Verfaillie. Earth observation satellite management. *Constraints*, 4(3) :293–299, 1999.

[3] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Constraints*, 4(1) :79–89, 1999.

[4] S. de Givry. Rapport de hdr, INRA UBIA Toulouse. 2011.

[5] M. Fontaine, S. Loudni, and P. Boizumault. Guiding vns with tree decomposition. In *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*, pages 505–512. IEEE, 2011.

[6] P. Hansen, N. Mladenovic, and D. Perez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4) :335–350, 2001.

[7] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *IJCAI'95*, pages 607–615, 1995.

[8] P. Jégou, S. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *CP*, pages 777–781, 2005.

[9] M. Kitching and F. Bacchus. Exploiting decomposition on constraint problems with high tree-width. In *IJCAI*, pages 525–531, 2009.

[10] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *IJCAI*, pages 239–244, 2003.

[11] S. Loudni and P. Boizumault. Combining VNS with constraint programming for solving anytime optimization problems. *EJOR*, 191 :705–735, 2008.

[12] R. Marinescu and R. Dechter. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artif. Intell.*, 173(16-17) :1457–1491, 2009.

[13] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers And Operations Research*, 24 :1097–1100, 1997.

[14] B. Neveu and G. Trombettoni. Incop : An open library for incomplete combinatorial optimization. In *Principles and Practice of Constraint Programming–CP 2003*, pages 909–913. Springer, 2003.

[15] B. Neveu, G. Trombettoni, and F. Glover. ID Walk : A candidate list strategy with a simple diversification device. In *CP'04, LNCS 3258*, pages 423–437, 2004.

[16] L. Perron, P. Shaw, and V. Furnon. Propagation guided large neighborhood search. *Principles and Practice of Constraint Programming–CP 2004*, pages 468–481, 2004.

[17] N. Robertson and P.D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms*, 7(3) :309–322, 1986.

[18] M. Sánchez, D. Allouche, S. de Givry, and T. Schiex. Russian doll search with tree decomposition. In *IJCAI*, pages 603–608, 2009.

[19] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *CP-98*.

[20] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3) :566–579, 1984.

[21] H. van Benthem. Graph : Generating radiolink frequency assignment problems heuristically, 1995.

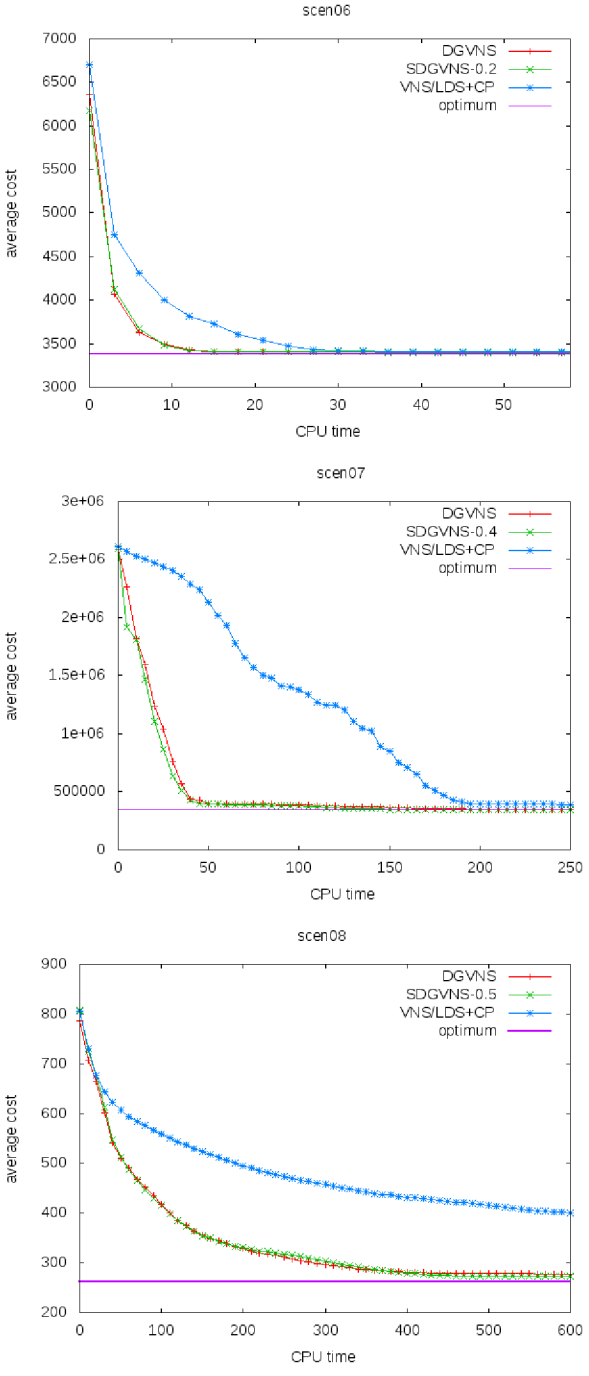
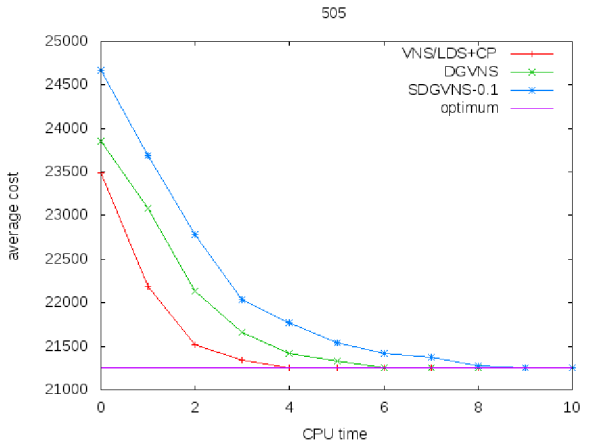
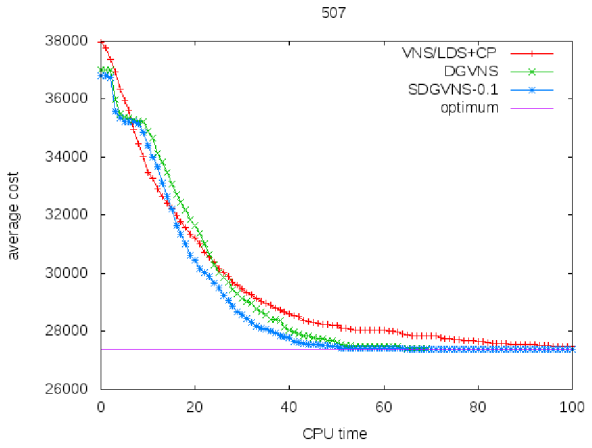


FIGURE 10 – Comparaison des profils de performance sur les instances RLFAP.





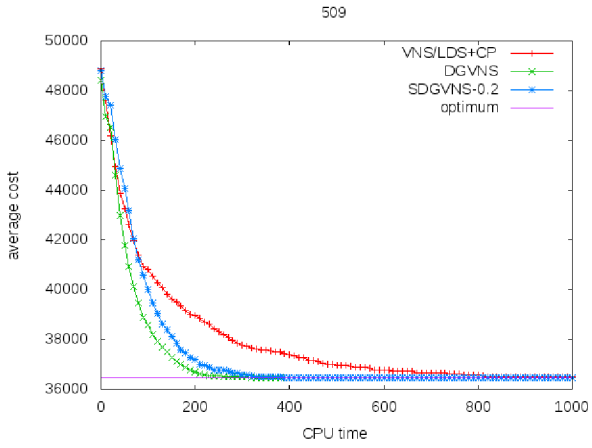
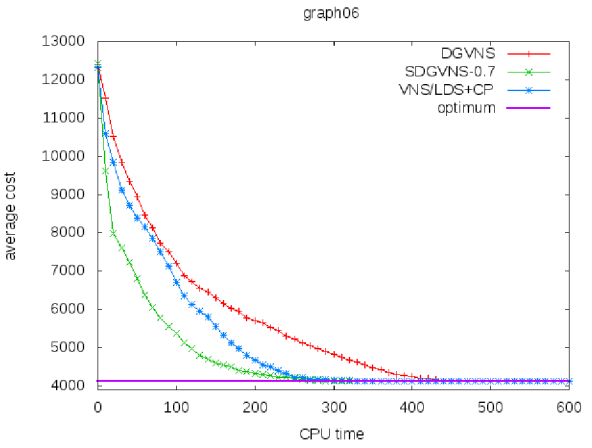
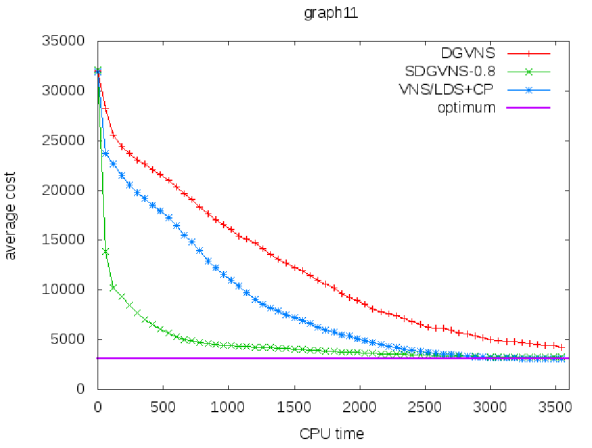


FIGURE 11 – Comparaison des profils de performance sur les instances SPOT5.





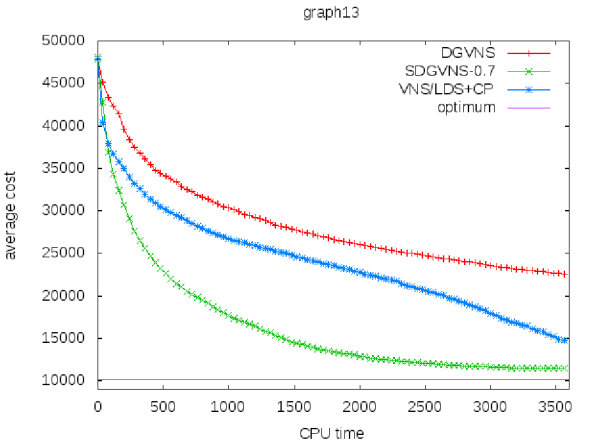


FIGURE 12 – Comparaison des profils de performance sur les instances GRAPH.

Optimisation énergétique de tables horaires de métros: une approche hybride

David Fournier[1 2] **Denis Mulard**[1] **François Fages**[2]

[1] General Electric Transportation ICS Delta
Tour Europlaza 28D5, 20 avenue André Prothin 92063 Paris La Défense Cedex (France)
[2] INRIA Rocquencourt Equipe Contraintes
Domaine de Voluceau Bâtiment 8 Rocquencourt BP 105 78153 Le Chesnay (France)
{david.fournier,francois.fages}@inria.fr denis.mulard@ge.com

Résumé

Sur une ligne de métro, les trains sont capables de générer de l'énergie en freinant. Cette énergie est disponible immédiatement dans le troisième rail et est perdue si aucun métro accélérant dans le voisinage ne peut la capter. En synchronisant les freinages et les accélérations il est donc possible de diminuer la consommation d'énergie globale du système.

Nous décrivons un modèle permettant de ne pas altérer la qualité de service en jouant de façon subtile sur les temps de stationnement des métros tout au long de leur voyage.

Un algorithme hybride génétique-linéaire a été implémenté pour résoudre ce problème et calculer la fonction de distribution d'énergie des métros producteurs.

Sur un exemple représentatif, la consommation d'énergie est ainsi diminuée de plus de 6%.

Abstract

In a metro line, metros are capable to generate electricity by braking. This energy, immediately available in the third rail, is lost if no metro in the neighborhood can consume it. Thus it is possible to decrease total energy consumption by synchronizing accelerations and brakings.

We describe a model which does not alter quality of service by subtly modifying dwell times.

A hybrid genetic/linear algorithm has been implemented to tackle this problem and compute the distribution of braking metros.

On a typical example, the savings are more than 6%.

1 Introduction

L'optimisation de la consommation d'énergie est une problématique majeure du XXI^{e} siècle et les métros n'y échappent pas. Plusieurs solutions matérielles, telles que les volants d'inertie ou les super-capacités ont été développées pour réduire les pertes. Cependant, ces solutions impliquent la mise en place de nouveaux équipements coûteux sur les métros qu'il est parfois difficile de justifier financièrement.

Parallèlement, des solutions logicielles se contentant de modifier certains paramètres ont vu le jour. Cependant, optimiser une table horaire est réputé difficile et il a été prouvé que certaines instances se rapportent à des problèmes NP-complets connus [2]. Les optimisations sont donc par essence des recherches d'optima locaux dès que les instances sont de grande taille. Albrecht [1] implémente un algorithme génétique et montre qu'il est possible de diminuer la consommation d'énergie et plus particulièrement les pics de puissance en utilisant le temps de réserve des métros. Cependant cette optimisation se fait en modifiant les allures des trains, ce qui peut être compliqué à implémenter en temps réel. Kim et al. [4] proposent quant à eux un modèle en programmation linéaire mixte en nombres entiers pour optimiser les départs des métros. Cette modélisation est intéressante mais est trop simplifiée et les modifications sont trop simples pour pouvoir correspondre à la réalité. Ces simplifications sont néanmoins obligatoires dans ce genre de modélisation du fait des temps de calculs. Enfin, Nasri et al. [5] montrent qu'il pourrait être possible de diminuer la consommation d'énergie en modifiant les temps de stationnement.

Dans cette article nous décrivons une méthode se focalisant sur la synchronisation logicielle des freinages et accélérations des métros en jouant sur les temps

de stationnement, variables facilement modifiables en temps réel. Pour cela, nous utiliserons la programmation linéaire hybridée à un algorithme génétique pour réduire la consommation globale du réseau plutôt que de seulement lisser les pics de puissance et ce de façon rapide et efficace.

2 Problématique

L'énergie consommée dans une ligne peut être diminuée en synchronisant les freinages et les accélérations des métros. En effet, le moteur électrique d'un métro qui freine se comporte comme un générateur en transformant son énergie cinétique en énergie électrique directement disponible dans le 3^e rail. Cette énergie est instantanée et se dissipe si aucun métro dans le voisinage ne peut l'absorber.

Un métro candidat ne peut de toute façon absorber qu'une partie de l'énergie générée par un métro au freinage. Le reste de cette énergie est dissipée dans le circuit électrique sous forme de chaleur.

La plupart des tables horaires ne tiennent pas compte des problématiques énergétiques. Elles ont été créées pour donner une qualité de service maximale aux usagers mais en tenant aussi compte, par exemple, des plages de travail des conducteurs, des heures de la journée ou de la période de l'année.

Il est bien souvent hors de question de modifier complètement une table horaire seulement pour prendre en compte des problématiques énergétiques. Il est cependant possible de modifier légèrement celles-ci pour inclure une optimisation de l'énergie consommée sur une période de temps donnée.

On cherchera ici à minimiser la consommation d'énergie d'une ligne de métro sur une tranche horaire précise en modifiant la table horaire hors-ligne.

3 Données

Le modèle est restreint à une ligne de métro simple (pas de fourche ou de boucle) composée de 31 stations dont deux terminus A et B. La totalité des métros effectue le trajet A vers B ou B vers A suivant leur orientation.

La table horaire, basée sur des données réelles, est un peu plus détaillée que celle donnée à l'usager. Outre les heures de départ en terminus et en station, elle se décompose en différents tableaux compilant entre autres :

- les temps de parcours entre chaque station
- les temps de stationnement à chaque station.

Le temps de stationnement représente le temps d'attente nominal d'un métro dans une station donnée. Ce temps peut être différent suivant les stations mais on considère ici que tous les métros ont toujours le même temps de stationnement pour une station donnée, quelle que soit l'heure de la journée.

Pour chaque intervalle de temps (1 seconde dans le modèle), on connaît aussi la position du métro (entre quelles stations il se trouve) et son énergie, positive s'il en consomme et négative s'il en génère. Contrairement aux données de temps qui sont réelles, les données d'énergie ont été créées en suivant toutefois des modèles d'énergie comme dans [4]. Les unités sont arbitraires : une valeur de 1 dans ce système équivaut à l'énergie consommée par un train accélérant à pleine puissance pendant 1 seconde.

Une matrice d'atténuation compile les dissipations d'énergie entre différents points du réseau. Par exemple que si un train génère de l'énergie au point a, un train au point b ne pourra absorber que $\tau\%$ de cette énergie, le reste étant dissipé dans la voie.

4 Modèle

4.1 Table horaire

L'objectif (1) est de minimiser la consommation d'énergie sur une période de temps donnée, donc de minimiser la somme des consommations d'énergie à chaque intervalle de temps. Si on prend T l'ensemble des intervalles de temps et y_t la consommation d'énergie du réseau à l'instant t on a donc comme fonction objectif :

$$\min \sum_{t \in T} y_t \tag{1}$$

Pour calculer y_t, on utilise un module de programmation linéaire. La minimisation de la fonction objectif, c'est-à-dire l'optimisation de la consommation d'énergie sur un horizon T se fait en modifiant les temps de stationnement pertinents sur cet horizon. Le fait de modifier les temps de stationnements sur l'horizon permet de décaler les horaires des trains par rapport aux autres et ainsi de mieux synchroniser les accélérations et freinages entre eux.

Le calcul des des temps de stationnements pertinents se fait d'abord en calculant l'ensemble des trains pertinents sur l'horizon comme suit :

Ensembles

- T : horizon de temps (intervalle).
- I : métros.
- S : stations.
- $\mathcal{I}_t$: métros roulant (pertinents) à l'instant $t \in T$.
- $\mathscr{I}_T$: métros pertinents sur l'horizon T.
- Δ_i : temps de stationnement de $i \in I$.
- $\mathcal{D}_{i,T}$: temps de stationnement pertinents de $i \in I$ sur l'intervalle T.

– $\mathscr{D}_T$: temps de stationnement pertinents sur l'intervalle T.

Paramètres

– D_i : départ au terminus initial de $i \in I$.
– $D_{d_{i,s}}$: instant d'arrivée de $i \in I$ à son point de stationnement à la station $s \in S$.
– L_i : temps de trajet initial de $i \in I$.
– T_0 : borne inf de T.
– T_{MAX} : borne sup de T.

Variables

– $d_{i,s}$: temps de stationnement de $i \in I$ à la station $s \in S$.

Modèle

$$\mathscr{I}_T = \bigcup_{t \in T} \mathcal{I}_t \tag{2}$$

avec $$\mathcal{I}_t = \{i \in I / D_i + L_i \geq t \geq D_i\} \tag{3}$$

(3) et (2) explicitent que les trains pertinents sont ceux qui roulent effectivement sur la voie au moins à un instant donné de l'intervalle de temps que l'on prend pour horizon.

Le calcul des trains pertinents sur l'horizon (2) permet de calculer les temps de stationnement pour chacun d'eux :

$$\mathscr{D}_T = \bigcup_{i \in \mathscr{I}_T} \mathcal{D}_i \tag{4}$$

avec $$\mathcal{D}_i = \{d_{i,s} \in \Delta_i / T_0 \leq D_{d_{i,s}} \leq T_{MAX}\} \tag{5}$$

Ce sont donc ces temps de stationnement $d_{i,s} \in \mathscr{D}_T$ que l'on modifiera dans l'algorithme génétique pour tenter de minimiser la fonction objectif.

4.2 Énergie instantanée

La modification des temps de stationnements entraîne une synchronisation différente des métros entre eux. Il faut donc calculer à chaque itération de l'algorithme la fonction objectif en chaque instant. Le problème est qu'il est difficile de savoir en général comment se comporte dans le 3^e rail, l'électricité produite par les trains freinant. [3] fait l'hypothèse que les métros qui freinent peuvent donner entièrement leur énergie à d'autres métros mais seulement s'ils font partie de la même sous-section électrique (portion de la ligne de métro).

On fait ici l'hypothèse que l'énergie se dissipe selon une loi de distance entre les trains compilée dans une matrice d'atténuation. On fait aussi l'hypothèse qu'un métro peut potentiellement donner son énergie à un métro à n'importe quel point de la ligne.

Cependant, on considère que l'énergie régénérée par les métros freinant se répartit de façon optimale parmi les métros accélérant, le modèle tente donc de minimiser la perte d'énergie par dissipation. Pour un intervalle de temps donné on a donc :

Ensembles

– I^+ : métros consommant de l'énergie.
– I^- : métros produisant de l'énergie.

Paramètres

– E_i^+ : énergie consommée par le métro $i \in I^+ (> 0)$.
– E_i^- : énergie produite par le métro $i \in I^- (< 0)$.
– $A_{i,j}$: dissipation par effet Joule entre le métro $i \in I^-$ et $j \in I^+$.

Variables

– $x_{i,j}$: part de l'énergie produite par $i \in I^-$ donnée à $j \in I^+$.

Modèle

minimize y (6)

subject to

$$\sum_{i}^{I^+} E_i^+ + \sum_{i}^{I^-} (E_i^- . \sum_{j}^{I^+} x_{i,j} . A_{i,j}) \leq y \tag{7}$$

$$\sum_{j}^{I^+} x_{i,j} \leq 1 \quad \forall i \in I^- \tag{8}$$

$$-x_{i,j} . E_i^- . A_{i,j} \leq E_j^+ \quad \forall i \in I^-, \forall j \in I^+ \tag{9}$$

$$x_{i,j} \geq 0 \quad \forall i \in I^-, \forall j \in I^+ \tag{10}$$

$$y \geq 0 \tag{11}$$

(6) est la fonction minimisation alors que (7) force celle-ci à être supérieure ou égale à la somme des consommations d'énergie moins la somme des énergies produites corrigées par un coefficient d'atténuation. (8) force chaque métro producteur à partager son énergie entre les métros consommateurs. (9) empêche un métro producteur de donner à un métro consommateur plus que ce qu'il peut accepter. (11) explicite qu'il ne peut pas y avoir d'énergie négative sur un intervalle de temps. Si l'énergie produite par les métros est supérieure à l'énergie consommée, alors ce surplus est perdu.

5 Algorithme génétique

La synchronisation des métros se fait en modifiant les temps de stationnement. Il est possible de modifier

légèrement ceux-ci sans altérer la qualité de service. Typiquement, un usager ne remarquera pas si son métro reste quelques secondes de plus ou de moins arrêté à une station. De plus, faire de légères modifications n'influence pas la solution finale en termes de faisabilité puisqu'on reste dans les intervalles acceptables de distance entre les métros par exemple.

Chaque individu de la population est représenté par une liste arrêts en station des différents métros roulant pendant la durée de l'intervalle que l'on veut optimiser. Cette liste est couplée aux temps de stationnement correspondants.

En partant des temps de stationnement initiaux, on crée une population de 100 individus dont les temps de stationnement sont tirés au hasard parmi un domaine prédéfini {-3s, 0s, +3s, +6s, +9s}. A chaque itération, les individus sont classés selon la qualité de leur fonction objectif, puis sélectionnés, croisés et mutés jusqu'à convergence.

6 Résultats

6.1 Temps de calcul

Le modèle a été testé sur un horizon d'une heure, correspondant à 3600 intervalles de temps d'une seconde, 29 métros et 495 temps de stationnement à optimiser. La valeur initiale de l'énergie totale consommée sur cette période est de 8504 u.a. Après 450 itérations, l'énergie totale est passée à 7939,4 u.a, soit une réduction de 6,6%.

Le temps de calcul sur un PC Linux équipé d'un processeur double cœur Intel Core 2 1,86GHz est de plus de 88h. Ce temps de calcul reflète un objectif d'optimisation maximum plutôt que de rapidité. Le but du modèle étant d'optimiser une table horaire hors-ligne, la contrainte est sur la qualité de la solution plutôt que sur le temps passé à la trouver.

6.2 Robustesse

Un réseau de métro réel est sujet à de multiples perturbations mineures qui peuvent affecter de quelques secondes les différents paramètres.

Pour vérifier la pertinence de l'optimisation, on a ajouté du « bruit » aléatoire sur les temps de stationnement optimisés pour quantifier la robustesse de la fonction objectif. Ce bruit consiste à modifier aléatoirement les temps de stationnement de $\pm\delta s$.

La table 1 compile ces résultats. On y voit que même avec un bruit de 6 secondes (correspondant à 2 intervalles de modification des temps de stationnement), la fonction objectif reste de bonne qualité par rapport à l'état initial puisque la réduction moyenne reste tout de même de 5,6%. Cela signifie que non seulement la solution optimisée est de bonne qualité mais que toutes les solutions voisines le sont aussi.

Bruit (s)	1	3	6
Moyenne sur 100 essais (u.a.)	7964,9	7995,7	8028,4
Economie (%)	6,3	6,0	5,6

TABLE 1 – Altération de la fonction objectif en fonction du bruit

7 Perspectives

Cette méthode de résolution pour l'optimisation de l'énergie consommée sur une ligne de métro semble prometteuse et mérite de plus amples recherches. Le choix des paramètres à optimiser est à approfondir. Il serait en effet intéressant de modifier les heures de départ ou les allures des métros pour optimiser encore plus finement.

On pourra essayer de comparer les résultats de cette méthode avec un codage en programmation linéaire mixte par nombres entiers.

En abaissant les temps de calcul, cette méthode devra permettre d'effectuer des optimisations en temps réel, en particulier quand il est question d'optimiser la consommation d'énergie après des incidents en ligne.

Références

[1] T. Albrecht. Reducing power peaks and energy consumption in rail transit systems by simultanoeus métro running time control. *Computers in Railways IX*, 2004.

[2] E. Bampas, G. Kaouri, M. Lampis, and A. Pagourtzis. Periodic metro scheduling. *ATMOS 2006*, 2006.

[3] K.M. Kim, K.T. Kim, and M.S. Han. A model and approaches for synchronized energy saving in timetabling. *WCRR 2011*, Mai 2011.

[4] Kyung min Kim, Suk mun Oh, and Moonseob Han. A mathematical approach for reducing the maximum traction energy : The case of korean mrt trains. *IMECS 2010*, Mars 2010.

[5] A. Nasri, M. Fekri Moghadam, and H. Mokhtari. Timetable optimization for maximum usage of regenerative energy of braking in electrical railway systems. *SPEEDAM 2010*, 2010.

Une recherche locale dirigée par l'analyse de conflits pour la satisfiabilité *

Djamal Habet **Donia Toumi**
LSIS – CNRS UMR 7296
Université Aix-Marseille
{Djamal.Habet, Donia.Toumi}@lsis.org

Résumé

Cet article présente un algorithme de recherche locale guidée par l'analyse de conflits pour résoudre le problème SAT. L'usage d'une telle analyse, permettrait d'exploiter les dépendances entre les variables particulièrement présentes dans des instances structurées et d'accroître l'effet de la propagation unitaire. Les premiers résultats expérimentaux sont prometteurs.

1 Introduction

Le problème de satisfiabilité (SAT) consiste à décider si une formule booléenne sous la forme normale conjonctive est satisfiable. SAT est l'un des problèmes NP-complets les plus étudiés à cause de son importance aussi bien sur le plan théorique que pratique.

L'intérêt des mécanismes de l'analyse des conflits et de l'apprentissage des clauses dans la résolution du problème SAT a été notamment démontré dans les démonstrateurs complets (par exemple dans [3, 7]). Cependant, peu de travaux exploitent de telles techniques dans le cadre de la recherche locale [1, 4]. Par ailleurs, des instances SAT, en particulier structurées, peuvent contenir des dépendances entre les variables dont la prise en compte permet d'améliorer leur résolution. Nous présentons dans ce papier une nouvelle intégration de l'analyse des échecs dans une recherche locale qui permet d'inclure la propagation unitaire sur une interprétation complète en cours de réparation captant ainsi d'éventuelles relations entre variables. Cette intégration introduit aussi un caractère tabou dans la recherche prévenant ainsi l'occurrence de cycles lors de l'exploration de l'espace de recherche. Les premiers résultats expérimentaux nous semblent prometteurs.

*Avec le soutien du projet ANR-08-BLAN-0289-04

Ce papier est organisé comme suit : dans la section 2 après la donnée de quelques définitions et notations, nous effectuons un rappel sur la recherche locale pour SAT, par la présentation de Walksat [8] qui est utilisé comme base dans notre approche, et un bref rappel du principe de l'analyse de conflits. La section 3 est consacrée à la description de notre approche, en décrivant les éléments qui la composent puis en expliquant son principe de fonctionnement. Les sections 4 et 5 présentent des travaux proches et des premiers résultats expérimentaux, avant de conclure dans la section 6.

2 Préliminaires

2.1 Définitions et notations

Une instance $\mathcal{F}$ du problème de satisfiabilité (SAT) est définie par la paire $\mathcal{F} = (\mathcal{X}, \mathcal{C})$, tels que $\mathcal{X} = \{x_1, x_2, \cdots x_n\}$ est un ensemble de variables booléennes (leurs valeurs appartiennent à l'ensemble $\{vrai,\ faux\}$) et $\mathcal{C} = \{c_1, c_2, \cdots c_m\}$ est un ensemble de clauses. Une clause $c_i \in \mathcal{C}$ est une disjonction finie de littéraux et un littéral est soit une variable (x_i) ou sa négation ($\neg x_i$). Une clause est aussi représentée par l'ensemble des littéraux y apparaissant. Pour un littéral donné l, $var(l) = \{x_i |\ l = x_i\ ou\ l = \neg x_i\}$ correspond à l'ensemble singleton de la variable sur laquelle porte l. De plus, pour une clause donnée c_j, $var(c_j) = \cup_{l\ \in\ c_j} var(l)$ est l'ensemble des variables apparaissant dans c. La clause vide est toujours fausse et notée par $\square$. Une interprétation $\mathcal{I}$ des variables de $\mathcal{F}$ est définie par un ensemble de littéraux, tel que $\forall\{l_1,\ l_2\} \subseteq \mathcal{I},\ var(l_1) \neq var(l_2)$. Une variable qui apparaît positivement (resp. négativement) dans $\mathcal{I}$ signifie qu'elle est fixée à $vrai$ (resp. à $faux$). $\mathcal{I}$ est dite partielle si $|\mathcal{I}| < n$ et complète sinon. Un modèle de $\mathcal{F}$

est une interprétation complète qui satisfait toutes les clauses de $\mathcal{F}$. Enfin, le problème de satisfiabilité (SAT) consiste à tester si $\mathcal{F}$ possède un modèle. Si c'est le cas alors $\mathcal{F}$ est dite satisfiable, sinon $\mathcal{F}$ est insatisfiable.

2.2 La recherche locale pour SAT

La description qui suit se limite à un rappel de l'algorithme Walksat [8] utilisé comme base dans cet article. Partant d'une configuration complète $\mathcal{I}_c$ falsifiant certaines clauses d'une instance $\mathcal{F}$ donnée, Walksat tente de minimiser le nombre de clauses falsifiées dans l'espoir d'atteindre un modèle. En effet, à chaque étape, Walksat tente de réparer $\mathcal{I}_c$ en inversant la valeur d'une variable (flip) apparaissant dans une clause falsifiée c_i, choisie aléatoirement. Pour toute variable $x_j \in var(c_i)$, une fonction *score* calcule le nombre de clauses qui seront falsifiées en cas de flip de x_j. Ces scores servent à choisir la variable à flipper. Ainsi, s'il existe une variable à score nul alors une telle variable est choisie (descente). Sinon, la recherche se trouve dans un minimum local. Dans ce cas, avec une probabilité p (*noise probability*), la variable avec le plus petit score est sélectionnée et avec une probabilité $1-p$ une variable est sélectionnée aléatoirement dans c_i (*marche aléatoire*). Etant incomplet, Walksat ne garantit pas la construction d'un modèle pour les instances satisfiables et ne peut prouver l'inconsistance des instances insatisfiables.

2.3 Analyse de conflits et graphe d'implications

L'analyse de conflits dans le cadre de la satisfiabilité a été introduite dans [7]. Elle est exploitée dans une recherche complète et arborescente, travaillant donc sur des interprétations partielles. Dans cette dernière, certaines variables sont fixées par décision et d'autres sont le résultat de la propagation des clauses unitaires induites par l'interprétation successive des variables. L'analyse des conflits est conduite lors de l'occurrence d'un échec correspondant à la falsification d'une clause. Dans ce cas, cette analyse identifie les littéraux (les faits) responsables de cet échec (conflit). Pour éviter que la recherche visite à nouveau la partie de l'espace de recherche contenant ce conflit, un *nogood* est ajouté sous forme d'une clause, dite *assertive*, aux clauses initiales de l'instance traitée. De plus, cette clause assertive permet d'augmenter l'effet de la propagation unitaire.

L'analyse de conflits nécessite la définition d'un graphe d'implications sur une interprétation partielle $\mathcal{I}_p$. Ce graphe est acyclique où les sommets sont des littéraux de $\mathcal{I}_p$ et il existe un arc d'un sommet l_i vers un sommet l_j si l'affectation de $var(l_i)$ implique celle de $var(l_j)$ par propagation unitaire. Ainsi, les variables de décision ne possèdent pas d'arcs entrants. Aussi, tout sommet (littéral) est étiqueté par la clause (devenue) unitaire ayant provoquée sa propagation et à tout littéral est associé un niveau de décision.

L'analyse de conflits exploite ce graphe d'implications (le plus souvent) par le mécanisme *Unique Implication Point* (UIP) [9]. Le nogood (clause assertive) responsable de l'échec est généré en effectuant des résolvantes entre les clauses participant à la génération du conflit. Au même temps, un niveau de retour-arrière non-chronologique est défini.

3 Notre contribution : AcWalk

3.1 Motivations

L'une des faiblesses de la recherche locale est son manque d'exploitation des dépendances entre les variables. Il est reconnu qu'une telle exploitation permet d'augmenter l'efficacité des démonstrateurs SAT, qu'ils soient complets ou à base d'une recherche locale. La prise en compte de ces dépendances passe notamment par la propagation unitaire, technique qui est difficile à mettre en œuvre dans une recherche locale qui, comme vu précédemment, agit sur des configurations complètes. Un autre problème "classique" de la recherche locale est sa tendance, dans certains cas, à se retrouver piéger dans une zone de l'espace de recherche sans pouvoir la quitter. L'une des solutions à ce problème est d'appliquer la métaheuristique tabou [5]. Nous verrons dans la suite la mise en place de cette dernière dans notre approche. Par ailleurs, l'intérêt de l'analyse de conflits et l'apprentissage des clauses (nogoods) n'est plus à démontrer. Ces techniques sont des composantes indispensables dans les démonstrateurs SAT actuels. Toutefois, on ne trouve dans la littérature que très peu de travaux sur l'intégration de ces mécanismes de l'analyse des conflits dans une recherche locale (par exemple [1, 4]).

La section suivante décrit le principe de notre algorithme, que nous appelons AcWalk, qui tente de répondre aux points cités ci-dessus.

3.2 Composantes d'AcWalk

Comme exposé précédemment, l'une des difficultés liées à la recherche locale est l'incapacité de celle-ci à intégrer la propagation unitaire lors de la recherche. Par ailleurs, l'analyse de conflits nécessite la construction d'un graphe d'implication, sur la base des propagations unitaires, à partir d'une interprétation partielle ce qui est incompatible avec l'usage d'interprétations complètes pour une recherche locale. D'où l'idée de maintenir à la fois une interprétation partielle $\mathcal{I}_p$

(servant pour la construction du graphe d'implications) et une interprétation complète $\mathcal{I}_c$ (servant pour la recherche locale). Tout le long de la recherche, la relation $\mathcal{I}_p \subseteq \mathcal{I}_c$ sera maintenue comme expliqué ci-après.

3.2.1 Construction de $\mathcal{I}_p$ et $\mathcal{I}_c$

Dans ce papier, l'algorithme de recherche locale que nous utilisons est Walksat qui effectue à chaque itération un flip sur une variable x_i selon la fonction score rappelée dans la section 2.2. Deux cas se distinguent, soit $score(x_i) = 0$ donc on est dans le cas d'une descente et dans le cas contraire ($score(x_i) > 0$) un minimum local est atteint.

Définition 1. (Clause conflit) Soit $\mathcal{F}$ une instance SAT et une interprétation complète $\mathcal{I}_c$ sur les variables de $\mathcal{F}$ et soit c_k une clause falsifiée par $\mathcal{I}_c$. c_k est dite clause conflit si $\forall x_j \in var(c_k),\ score(x_j) > 0$.

Autrement dit, une clause conflit est une clause falsifiée où toutes les variables ont des scores négatifs. Nous définissons ainsi une variable conflit comme suit :

Définition 2. (Variable conflit) Soit $\mathcal{F}$ une instance SAT et une interprétation complète $\mathcal{I}_c$ sur les variables de $\mathcal{F}$ et soit c_k une clause falsifiée par $\mathcal{I}_c$. Si c_k est une clause conflit alors toute variable de $var(c_k)$ est dite variable conflit.

Lors d'un flip par Walksat d'une variable conflit x_i dans $\mathcal{I}_c$, le littéral correspondant à ce flip est ajouté à $\mathcal{I}_p$ ainsi que les propagations unitaires obtenues par l'application de $\mathcal{I}_p$ à $\mathcal{F}$. Le résultat de ces propagations est aussi appliquée à $\mathcal{I}_c$ permettant ainsi à la recherche locale de tirer profit de cette opération et de capturer certaines dépendances hypothétiques entre les variables. Avec une telle construction, à chaque flip d'une variable conflit, d'autres variables peuvent être aussi flippées. Aussi, on a bien $\mathcal{I}_p \subseteq \mathcal{I}_c$.

Par opposition à une variable conflit, notons que si une variable x_j apparait dans une clause falsifiée tel que $score(x_j) = 0$, le flip de cette variable ne produira aucun conflit si elle venait à être ajoutée à $\mathcal{I}_p$ et ne provoquera ainsi aucune analyse de conflits.

3.2.2 Construction et exploitation du graphe d'implications

Le graphe d'implications est construit sur la base des variables conflits. Ainsi, à chaque flip d'une telle variable, le littéral correspondant est ajouté à $\mathcal{I}_p$ et le graphe d'implication est mis à jour, en l'augmentant par les sommets et les arcs correspondant à ce flip ainsi que les littéraux propagés et leurs causes.

Intéressons nous maintenant à son exploitation : durant l'extension du graphe d'implication, un conflit peut se produire sur la base de l'interprétation $\mathcal{I}_p$. Dans ce cas, une analyse de conflits est effectuée comme décrit dans la section 2.3, où le niveau de décision de chaque littéral dans $\mathcal{I}_p$ correspondant à l'itération de la recherche locale ayant provoqué son ajout à $\mathcal{I}_p$. Plus précisément, lors d'un conflit un nogood (clause assertive) est ajoutée à la base des clauses et un niveau de retour-arrière it est fourni. Il s'en suit la suppression de $\mathcal{I}_p$ de tout les littéraux ajoutés depuis l'itération it. Après cette opération, l'inclusion de $\mathcal{I}_p$ dans $\mathcal{I}_c$ est toujours vérifiée.

Notons que dans certains cas, la clause assertive peut-être vide ce qui conduit à prouver l'insatisfiabilité de l'instance SAT traitée. Cela confère à cette recherche locale une capacité qu'elle ne possédait pas de fait de son caractère incomplet.

3.2.3 Gestion tabou

Lors d'une nouvelle réparation par Walksat, ce dernier se restreint à choisir une variable dans une clause falsifiée, dont aucun littéral correspondant n'appartient à $\mathcal{I}_p$. Ce qui revient à utiliser $\mathcal{I}_p$ comme une liste tabou et la durée tabou de chaque variable (littéral) est dynamique : tant que ce littéral n'est pas effacé de $\mathcal{I}_p$ à la suite d'une analyse de conflits, la variable correspondante reste tabou. La volonté de marquer ce caractère tabou vient du fait que les valeurs des variables dans $\mathcal{I}_p$ refléteraient une certaine dépendance entre les variables qu'il conviendrait de maintenir. Un procédé comparable a été utilisé aussi dans [2, 6]. Toutefois dans notre approche, on applique un critère d'aspiration qui permet de lever le caractère tabou d'une variable x_i si le flip de celle-ci permet de réaliser une descente, autrement dit $score(x_i) = 0$. A la même occasion, on lève le caractère tabou de toutes les variables flippées ou propagées depuis le flip de x_i en supprimant de $\mathcal{I}_p$ les littéraux correspondant (en conséquence $\mathcal{I}_p \subseteq \mathcal{I}_c$ reste vérifiée) et en mettant à jour le graphe d'implications.

3.2.4 AcWalk résumé

L'algorithme AcWalk résultant est un algorithme de recherche locale basé sur Walksat. Il a pour objectif de tirer profit de la propagation unitaire et de l'analyse de conflits, et peut même prouver dans certains cas l'insatisfiabilité de l'instance. Cette capacité reste toutefois secondaire dans la pratique. AcWalk maintient tout le long de la recherche deux interprétations : une partielle ($\mathcal{I}_p$ initialisée à $\emptyset$) et une autre complète ($\mathcal{I}_c$ initialisée

aléatoirementt). Tant qu'un modèle n'est pas trouvé dans le cas de la satisfiabilité (ou que la clause assertive n'est pas vide, dans le cas d'une instance insatisfiable), une variable à flipper est choisie selon les critères évoqués et un graphe d'implications est maintenue à jour tout en appliquant l'effet des propagations sur les variables de l'interprétation complète. La configuration partielle sert aussi de liste tabou qui est toutefois remise en cause dans le cas d'une descente. A chaque essai, AcWalk effectue un nombre fixé de réparations sur une configuration initiale générée aléatoirement et ce processus est répété selon un critère d'arrêt prédéfini (limitation de temps, nombre maximum d'essai autorisés ...).

4 Travaux connexes

Cette courte étude bibliographique se limite à l'intégration de l'analyse des conflits dans une recherche locale. Dans [1], les auteurs proposent un algorithme de recherche locale CDLS (aussi basé sur Walksat) utilisant l'analyse de conflits que dans le but de s'échapper des minima locaux. Dans ce cas, une interprétation partielle est dérivée à partir de l'interprétation complète courante. Elle sert à la construction d'un graphe de conflits qui est analysé par la suite. Ainsi, tandis qu'AcWalk maintient un graphe d'implications et une interprétation partielle (et complète) tout le long de la recherche, CDLS reconstruit un graphe conflit à chaque minimum local. Dans [2], une méthode SatHys est définie par l'hybridation d'une recherche locale et un démonstrateur complet de type minisat [3]. Ces deux recherches travaillent alternativement selon des critères bien définis et partagent certaines informations. Cette hybridation constitue une première différence majeure avec AcWalk qui est un algorithme de recherche locale. Une autre distinction est la gestion de la liste tabou dans Sathys, qui n'introduit pas un mécanisme d'aspiration.

5 Résultats expérimentaux

Une étude expérimentale préliminaire a été effectuée sur une première implémentation d'AcWalk. Les tests sont effectués sur des instances structurées et satisfiables issues principalement de SATLIB. Les instances aléatoires présentant peu de relations structurelles entre les variables, elles sont de ce fait hors du cadre de nos objectifs. Nous comparons AcWalk (codé en C/C++) avec CDLS et Walksat. Le choix de CDLS est motivé par le fait qu'il soit (à notre connaissance) le seul algorithme de recherche locale connu exploitant l'analyse de conflits. AcWalk étant basé sur Walkat, la comparaison avec celui-ci est pertinente afin de mesurer l'apport de notre approche. Les tests sont réalisés sur des lames de calcul avec 2 processeurs Intel Xeon 2.4 Ghz et 24 GO de mémoire vive. Les trois algorithmes sont testés avec un nombre maximum de réparations de 10^6 par essai. Les essais sont répétés pendant un temps limite de 1800 secondes. Chaque algorithme est lancé 10 fois et nous mesurons le taux de réussite (t dans $[0,\ 1]$), le temps moyen ($tmoy$) et médian ($tmed$) en secondes sur les exécutions ayant abouties à la construction d'un modèle. Dans tout les tests, la probabilité p pour la marche aléatoire est fixée à 0.5.

Comparé à Walksat, les gains sur ces instances sont très significatifs, que ce soit en temps d'exécution ou en taux de réussite. Nous observons aussi des meilleurs résultats par rapport à CDLS. Même si d'autres tests sont nécessaires afin de positionner notre approche par rapport à d'autres démonstrateurs incomplets, ces premiers résultats montrent toutefois la pertinence de notre approche.

6 Conclusion

Pour mieux résoudre des instances structurées, nous avons présenté une nouvelle approche permettant d'intégrer l'analyse des conflits et l'apprentissage de clauses dans un algorithme de recherche locale pour SAT et les premiers résultats sont encourageants. Parmi les points à améliorer, la gestion des clauses apprises est l'un des points essentiels. En effet, dans la version actuelle d'AcWalk, les clauses asservies sont utilisées pour augmenter l'effet de propagation unitaire et comme nogoods. Le calcul des scores des variables ne se fait que sur les clauses originelles (non apprises). Ce choix est dû à une observation empirique qui a montrée qu'il n'était pas judicieux (du moins d'un point de vue efficacité pratique) d'inclure ces clauses dans le choix de la variable à flipper. Aussi, le nombre de clauses apprises doit être maitrisé afin d'éviter un accroissement trop important, provoquant ainsi une perte significative de l'efficacité pratique de notre solveur. Les techniques utilisées dans le cadre d'une recherche de type minisat sont une première solution mais qu'il est nécessaire d'affiner dans le cadre de la recherche locale.

Par ailleurs, une gestion plus étudiée de la liste tabou doit-être menée. En effet, telle que présentée, cette gestion peut-être agressive (lors de l'application du critère d'aspiration) si la variable à flipper figure parmi les premières à être insérées dans l'interprétation $\mathcal{I}_p$. Une première approche consisterait à introduire un second critère lors de choix de la variable flipper qui consiste à prendre en compte l'âge de la variable, qui correspond à sa date d'insertion dans $\mathcal{I}_p$.

Instances	**Walksat** ($t/tmoy/tmed$)	**CDLS** ($t/tmoy/tmed$)	**AcWalk** ($t/tmoy/tmed$)
bmc-ibm-1	0/-/-	1/50.9/49.74	**1/1.77/1.75**
bmc-ibm-2	0/-/-	1 /0.03/0.03	1/0.01/0.01
bmc-ibm-3	0/-/-	1/101.4/101.0	**1/6.33/6.23**
bmc-ibm-4	0/-/-	1/33.0/33.0	**1/1.50/1.17**
bmc-ibm-5	0/-/-	1/1.60/1.65	1/0.09/0.09
bmc-ibm-6	0/-/-	1/84.78/80.21	**1/2.34/2.29**
bmc-ibm-7	0/-/-	1/0.04/0.04	1/0.03/0.03
bmc-ibm-10	0/-/-	1/36.9/35.5	1/16.51/15.90
bmc-ibm-11	0/-/-	0.8/1451.8/1437.8	**1/11,66/10,98**
bmc-ibm-12	0/-/-	0 /- /-	**1/305.86/292.24**
bmc-ibm-13	0/-/-	0.7/1185.6/1314.1	**1/118.91/104.48**
bmc-gal-8	0/-/-	1/492.7/500.4	**1/7.47/7.41**
bmc-gal-9	0/-/-	1/715.8/725.0	**1/13.22/12.98**
qg1-07	1/24.4/18.6	1/0.18/0.15	1/0.03/0.03
qg1-08	0/-/-	1/472.76/398.37	**1/11.63/5.99**
qg2-07	1/18.4/6.5	1/0.17/0.15	1/0.02/0.02
qg2-08	0/-/-	0.6/819.2/836.9	**1/69.51/42.18**
qg3-08	1/9.5/7.7	1/0.08/0.05	1/0.002/0.002
qg4-09	1/63.2/14.0	1/0.78/0.21	1/0.007/0.004
qg5-11	0/-/-	1/0.47/0.27	1/0.22/ 0.14
qg6-09	0-/-	1/0.02/0.02	1/0.006/0.008
qg7-09	0.1/1078.9/1078.9	1/0.01/0.01	1/0.00/0,004
qg7-13	0/-/-	1/169.2/98.9	**1/12,8/6,87**
bw_large.c	0.3/926.7/941.4	1/4.17/ 4.14	1/0,40/0.38
bw_large.d	0-/-/-	1/138.5/121.2	**1/23.87/25.28**
par16-1	0-/-/-	1/96.0/ 28.1	**1/16.29/10.30**
par16-2	0-/-/-	1/176.1/122.2	**1/54.09/33.75**
par16-3	0-/-/–	1/180.3 /168.9	**1/19.27/16.98**
par16-4	0-/-/-	1/119.0/123.8	**1/13.02/7.57**
par16-5	0-/-/-	1/124.9/108.2	**1/29.78/31.79**
par16-1-c	0-/-/-	1/25.4/23.1	**1/39.86/30.21**
par16-2-c	0-/-/-	**1/78.4/46.2**	1/82,12/63,74
par16-3-c	0-/-/-	1/60.05/43.02	**1/0,42/0,06**
par16-4-c	0-/-/-	**1/31.5/26.5**	1/33,62/14,94
par16-5-c	0-/-/-	1/81.2/84.4	**1/34,29/26,68**
vmpc_24	0.1/1184.1/1184.1	0.7/504.53/447.86	**1/227.21/55.23**
vmpc_25	0-/-/-	0.7/280.28/302.86	**1/313.58/179.54**
vmpc_26	0-/-/-	0/-/-	**0.9/469.99/394.66**
vmpc_28	0-/-/-	0/-/-	**0.5/476.01/221.78**
vmpc_29	0-/-/-	0/-/-	**0.3/753.80/463.35**
vmpc_30	0-/-/-	0/-/-	**0.2/507.74/507.74**
vmpc_33	0-/-/-	0/-/-	0/-/-
vmpc_34	0-/-/-	0/-/-	0/-/-

TABLE 1 – Premiers résultats comparatifs

Références

[1] G. Audemard, JM. Lagniez, B. Mazure, and L. Sais. Analyse de conflits dans le cadre de la recherche locale. In *JFPC*, pages 215–224, 2009.

[2] G. Audemard, JM. Lagniez, B. Mazure, and L. Sais. Boosting local search thanks to cdcl. In *LPAR*, pages 474–488, 2010.

[3] N. Eén and N. Sörensson. An extensible sat-solver. In *SAT*, pages 502–518, 2003.

[4] Hai Fang. Complete local search for propositional satisfiability. In *AAAI*, pages 161–166, 2004.

[5] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[6] L. Kroc, A. Sabharwal, CP. Gomes, and B. Selman. Integrating systematic and local search paradigms : A new strategy for maxsat. In *IJCAI*, pages 544–551, 2009.

[7] JP. Marques-Silva and KA. Sakallah. Grasp : A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5) :506–521, 1999.

[8] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *AAAI*, pages 321–326, 1997.

[9] L. Zhang, CF. Madigan, and MH. Moskewicz. Efficient conflict driven learning in a boolean satisfiability solver. In *ICCAD*, pages 279–285, 2001.

Algorithme optimal d'arc-consistance pour une séquence de contraintes AtMost avec cardinalité

Emmanuel Hebrard[1,2] **Marie-José Huguet**[1,3] **Mohamed Siala**[1,3]

[1] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[2] Univ de Toulouse, LAAS, F-31400 Toulouse, France
[3] Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
{hebrard, huguet, siala}@laas.fr

Résumé

La contrainte AtMostSeqCard est la conjonction entre une contrainte de cardinalité sur une séquence de n variables et de $n - q + 1$ contraintes AtMost u sur toutes sous-séquences de variables de taille q. Elle se retrouve en particuliers dans des problèmes de type car-sequencing et crew-rostering. Une adaptation de deux algorithmes conçus pour la contrainte AmongSeq afin de traiter la contrainte AtMostSeqCard a été proposée dans [18]. Ces algorithmes ont une complexité temporelle respective au pire en $O(2^{q-1}n)$ et $O(n^3)$. Dans [10], un autre algorithme a été adapté de manière similaire pour la contrainte AtMostSeqCard avec une complexité temporelle au pire en $O(n^2 \log n)$. Cet article présente un algorithme réalisant l'arc-consistance de la contrainte AtMostSeqCard avec une complexité temporelle au pire en $O(n)$ (et donc optimal). Enfin, des expérimentations sont présentées pour évaluer l'efficacité de cet algorithme sur des problèmes de car-sequencing et de crew-rostering.

1 Introduction et état de l'art

Cet article s'intéresse à l'existence de restrictions sur des séquences de décision : certaines séquences peuvent être autorisées ou préférées et d'autres interdites. Par exemple, dans les problèmes de confection d'horaires pour des équipes (crew-rostering), il est, souvent interdit de positionner la même équipe sur un travail de soirée ou de nuit puis de la placer le lendemain sur un travail en matinée.

Les contraintes Regular [12] et cost-Regular [7] permettent d'exprimer des restrictions quelconques sur des séquences de décision. Toutefois, dans des cas particuliers, il existe des algorithmes plus efficaces. Par exemple, des algorithmes de filtrage ont été proposés pour la contrainte AmongSeq [6, 10, 19, 18]. Cette contrainte vérifie que toutes les sous-séquences de taille q contiennent au minimum l et au maximum u valeurs prises dans un ensemble v et est très employée pour modéliser des problèmes de car-sequencing et de crew-rostering. Mais, les contraintes de ces deux familles de problèmes ne se limitent pas uniquement à cette définition. En effet, souvent, il n'y a pas de borne inférieure sur le nombre de valeurs ($l = 0$) ; de plus, le nombre de valeurs prises dans l'ensemble v est généralement contraint par une demande globale.

Cet article étudie la contrainte AtMostSeqCard qui répond à ce besoin. Cette contrainte, impliquant n variables $x_1, \ldots, x_n$, assure que, dans chaque sous-séquence de longueur q, pas plus de u choix sont fixés à une valeur prise dans l'ensemble v, et que sur toute la longueur de la séquence, il y a exactement d choix fixés à une valeur prise dans v. Dans le cas de problèmes de car-sequencing, cette contrainte permet d'exprimer que, pour une option donnée, dans toute sous-séquence de taille q, il ne peut y avoir plus de u classes de véhicules avec cette option et que le nombre total de véhicules ayant cette option vaut exactement d. Dans le cas de problèmes de crew-rostering, cette contrainte permet de traduire, par exemple, qu'un employé doit disposer d'au minimum 16h de pause entre deux périodes de travail de 8h ou qu'une semaine de travail ne peut dépasser 40h, tout en fixant un nombre total d'heures devant être travaillées sur la période de planification.

Un problème de Satisfaction de Contraintes (CSP) est défini par un triplet $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ où $\mathcal{X}$ représente l'ensemble des variables, $\mathcal{D}$ l'ensemble des domaines finis pour chacune des variables et $\mathcal{C}$ l'ensemble des contraintes qui spécifient les combinaisons de valeurs autorisés sur des sous-ensembles de variables. On suppose que $\mathcal{D}(x) \subset \mathbb{Z}$ pour tout $x \in \mathcal{X}$, et on note $\min(x)$ et $\max(x)$ les valeurs minimale et maximale, de $\mathcal{D}(x)$. Une instanciation

d'un ensemble de variables $\mathcal{X}$ est un tuple w tel que $w[i]$ représente la valeur affectée à la variable x_i. Une contrainte $C \in \mathcal{C}$ portant sur un ensemble de variables $\mathcal{X}$ caractérise une relation, i.e., un sous-ensemble de tuples du produit cartésien du domaine des variables de $\mathcal{X}$. Une instanciation w est dite consistante pour une contrainte C ssi elle appartient à la relation correspondante. Une contrainte C est dite *arc-consistante* (AC) ssi, pour toute valeur j de chaque variable x_i qu'elle met en jeu, il existe une instanciation consistante w telle que $w[i] = j$. Pour $x_i = j$, on appelle *support*, une telle instanciation w .

Cet article est organisé de la manière suivante : la section 2, présente un bref état de l'art sur les contraintes de séquence. La section 3 expose un algorithme en temps linéaire (et donc optimal) pour propager la contrainte ATMOSTSEQCARD. Des expérimentations sur des instances de car-sequencing et de crew-rostering sont données en section 4 avant une conclusion en section 5.

2 Les contraintes de SEQUENCE

Plusieurs variantes des contraintes de séquence sont passées en revue avant de motiver le besoin de la contrainte ATMOSTSEQCARD.

Soient v un ensemble d'entiers et l, u, q des entiers. Les contraintes de séquence s'expriment par des conjonctions de contraintes AMONG limitant, pour un ensemble de variables, le nombre d'occurrences d'un ensemble de valeurs.

Définition 2.1 $\text{AMONG}(l, u, [x_1, \ldots, x_q], v) \Leftrightarrow$

$$l \leq |\{i \mid x_i \in v\}| \leq u$$

2.1 Séquence de contraintes AMONG

La contrainte AMONGSEQ est une séquence de contraintes AMONG de largeur glissante q sur un vecteur de n variables.

Définition 2.2 $\text{AMONGSEQ}(l, u, q, [x_1, \ldots, x_n], v) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \text{AMONG}(l, u, [x_{i+1}, \ldots, x_{i+q}], v)$$

Elle a été proposée dans [2], et le premier algorithme (incomplet) pour propager cette contrainte a été proposé dans [1]. Par la suite, deux algorithmes complets ont été développés pour propager cette contrainte AMONGSEQ [19, 18]. Le premier algorithme utilise une reformulation basée sur la contrainte REGULAR et réalise l'AC avec une complexité temporelle en $O(2^{q-1}n)$. Le second un algorithme réalise l'AC avec une complexité temporelle en $O(n^3)$. Cet algorithme peut être étendu pour traiter tout ensemble de contraintes AMONG portant sur des variables consécutives (contrainte GEN-SEQUENCE) et a alors une complexité temporelle en $O(n^4)$ dans le pire des cas. Dans le cas qui nous intéresse, la complexité de cet algorithme est ramenée à $O(n^3)$ car ATMOSTSEQCARD se décompose en $O(n)$ AMONG. Puis, un algorithme de flot, réalisant l'AC en $O(n^2)$ a été proposé dans [10]. Cet algorithme a pu également être adapté, en utilisant des outils plus généraux de programmation linéaire, pour traiter la contrainte GEN-SEQUENCE avec une complexité temporelle au pire en $O(n^2 \log n)$.

2.2 Séquence de contraintes ATMOST

Malgré son intérêt, la contrainte AMONGSEQ ne modélise pas exactement le type de contraintes nécessaires pour le car-sequencing et le crew-rostering. En effet, souvent il n'y a pas de borne inférieure sur la cardinalité des sous-séquences, i.e., $l = 0$. AMONGSEQ est ainsi inutilement générale sur ce plan là. De plus, la contrainte de capacité est souvent associée à une exigence de cardinalité.

Par exemple, pour le car-sequencing, les classes de véhicules demandant une option donnée ne peuvent être groupées ensemble car la station de travail nécessaire ne peut réaliser consécutivement qu'une partie des véhicules (*at most* u véhicules dans toute sous-séquence de longueur q). Le nombre total d'occurrences des différentes classes est également connu, et se traduit en une contrainte de cardinalité globale plutôt qu'en termes de bornes inférieures sur chaque sous-séquence.

Pour le crew-rostering, les motifs sur les périodes de travail autorisées peuvent être complexes, et la contrainte REGULAR est souvent utilisée pour les modéliser. Cependant, travailler au plus u périodes sur q est un cas important. Si les journées sont divisées en 3 périodes de 8h, ATMOSTSEQ avec $u = 1$ et $q = 3$ traduit le fait qu'aucun employé ne peut travailler plus d'une période par jour *et* qu'il doit y avoir une pause de 24h avant de changer de période de travail. De plus, comme pour le car-sequencing, la borne inférieure sur le nombre de périodes travaillées est globale (et dépend ici du contrat de travail).

Ainsi, il faut souvent traiter une séquence de contraintes ATMOST définie par :

Définition 2.3 $\text{ATMOST}(u, [x_1, \ldots, x_q], v) \Leftrightarrow$

$$\text{AMONG}(0, u, [x_1, \ldots, x_q], v)$$

Définition 2.4 $\text{ATMOSTSEQ}(u, q, [x_1, \ldots, x_n], v) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \text{ATMOST}(u, [x_{i+1}, \ldots, x_{i+q}], v)$$

On peut noter qu'il est facile de maintenir l'AC de cette contrainte. En effet, la contrainte ATMOST est monotone, c'est-à-dire, que l'ensemble des supports pour la valeur 0 est un sur-ensemble strict de l'ensemble des supports pour la valeur 1. On en déduit que la contrainte ATMOSTSEQ est AC ssi chaque contrainte ATMOST est AC [4].

Un bon compromis entre la puissance de filtrage et la complexité peut être effectué en couplant les raisonnements sur le nombre total d'occurrences de valeurs dans l'ensemble v et sur la séquence de contraintes AtMost. [1] La contrainte AtMostSeqCard est donc définie par la conjonction entre la contrainte AtMostSeq et des contraintes de cardinalité sur le nombre total d'occurrences de chaque valeur de v :

Définition 2.5 $\text{AtMostSeqCard}(u, q, d, [x_1, \ldots, x_n], v) \Leftrightarrow$

$$\text{AtMostSeq}(u, q, d, [x_1, \ldots, x_n], v) \wedge |\{i \mid x_i \in v\}| = d$$

Les deux algorithmes AC proposés dans [19] ont été adaptés [18] pour établir l'AC sur la contrainte AtMostSeqCard. Premièrement, de la même manière que AmongSeq peut être écrite avec la contrainte Regular, AtMostSeqCard peut s'exprimer à l'aide de la contrainte cost-Regular dans laquelle le cout représente la demande globale et est augmenté sur les transitions ayant la valuation 1. Cette procédure a la même complexité au pire, i.e., $O(2^{q-1}n)$. Deuxièmement, une variante plus générale de l'algorithme polynomial (Gen-Sequence) est utilisée pour propager la contrainte AtMostSeqCard décomposée en une conjonction de contraintes Among de la manière suivante :

$$\begin{aligned}\text{AtMostSeqCard}(u, q, d, [x_1, \ldots, x_n], v) \quad &\Leftrightarrow \\ \bigwedge_{i=0}^{n-q} \text{Among}(0, u, q, [x_{i+1}, \ldots, x_{i+q}], v) \quad &\wedge \\ \text{Among}(d, d, [x_1, \ldots, x_n], v)\end{aligned}$$

Comme dans cette décomposition, la dernière contrainte Among représentant la cardinalité n'a pas les mêmes paramètres l et u que les autres, on obtient, en utilisant respectivement les algorithmes de van Hoeve et al. ou de Maher et al., une procédure en $O(n^3)$ ou en $O(n^2 \log n)$ pour établir l'AC de la contrainte AtMostSeqCard.

2.3 La contrainte « Global Sequencing »

Dans le cas particulier du car-sequencing, il n'y a pas seulement une cardinalité globale pour les valeurs de v, mais chaque valeur correspond à une classe de véhicules et est contraint par un nombre d'occurrences. Ainsi, Puget et Régin ont considéré la conjonction entre la contrainte AmongSeq et la contrainte Gcc. Soient c_l et c_u deux fonctions sur les entiers tels que $c_l(j) \leq c_u(j)\ \forall j$, et soit $\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}(x_i)$. La contrainte de cardinalité globale (Gcc) est définie de la manière suivante :

Définition 2.6 $\text{Gcc}(c_l, c_u, [x_1, \ldots, x_n]) \Leftrightarrow$

$$\bigwedge_{j \in \mathcal{D}} c_l(j) \leq |\{i \mid x_i = j\}| \leq c_u(j)$$

La contrainte « Global Sequencing » (Gsc) est définie par :

Définition 2.7 $\text{Gsc}(l, u, q, c_l, c_u, [x_1, \ldots, x_n], v) \Leftrightarrow$

$$\text{AmongSeq}(l, u, q, [x_1, .., x_n], v) \wedge \text{Gcc}(c_l, c_u, [x_1, .., x_n])$$

Les fonctions c_l et c_u sont définis de telle sorte que pour une valeur v, $c_l(v)$ et $c_u(v)$ correspondent au nombre d'occurrences de la classe correspondante d'un véhicule. Une reformulation de cette contrainte en un ensemble de contraintes Gcc a été introduite dans [15]. Cependant, effectuer l'AC de cette contrainte est NP-difficile [3]. En fait, réaliser la consistance de bornes l'est aussi. [2]

3 La contrainte AtMostSeqCard

Cette section, présente un algorithme de filtrage linéaire pour la contrainte AtMostSeqCard. Tout d'abord un algorithme glouton pour trouver un support avec une complexité temporelle en $O(nq)$ est exposé. Puis, on montre qu'avec deux appels à cette procédure il est possible d'assurer l'AC de la contrainte. Enfin, on montre que la complexité dans le pire cas peut se réduire à $O(n)$.

[18] et [10] ont fait remarquer qu'on pouvait considérer des variables booléennes et $v = \{1\}$, puisque la décomposition suivante des contraintes Among conserve le même niveau de consistance car elle est Berge-acyclique [6] :

$$\begin{aligned}\text{Among}(l, u, [x_1, \ldots, x_q], v) \quad &\Leftrightarrow \\ \bigwedge_{i=1}^{q} x_i \in v \leftrightarrow x'_i = 1 \quad &\wedge \quad l \leq \sum_{i=1}^{q} x'_i \leq u\end{aligned}$$

Par la suite, on considére la restriction ci-dessous de AtMostSeqCard à des variables booléennes et avec $v = \{1\}$:

Définition 3.1 $\text{AtMostSeqCard}(u, q, d, [x_1, \ldots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} (\sum_{l=1}^{q} x_{i+l} \leq u) \wedge (\sum_{i=1}^{n} x_i = d)$$

Soit w un n-tuple dans $\{0,1\}^n$, $w[i]$ représente le i^{ime} élément de w, $|w| = \sum_{i=1}^{n} w[i]$ sa cardinalité et $w[i:j]$ le $(|j-i|+1)$-tuple égal à w sur la sous-séquence $[x_i, \ldots, x_j]$.

On montre tout d'abord qu'on peut trouver un support, de manière gloutonne, en fixant à 1 les variables dans l'ordre lexicographique à chaque fois que cela est possible, c'est à dire en respectant la contrainte AtMostSeq. En faisant cela, on obtient une instanciation de cardinalité maximale qui peut être facilement modifiée en une instanciation de cardinalité d. L'Algorithme 1 présente cette

1. Choix de modélisation utilisé dans [18] pour le car-sequencing.

2. Note interne "Comics" (Nina Narodytska).

règle gloutonne de calcul d'une instanciation w maximisant la cardinalité de la séquence $(x_1, \ldots, x_n)$ en respectant la contrainte ATMOSTSEQ.

En ligne 1, le vecteur w est initialisé à la valeur minimale du domaine de chaque variable. Puis, à chaque étape $i \in [1, \ldots, n]$ de la boucle principale, la variable $c(j)$ représente la cardinalité de la j^{eme} sous-séquence impliquant la variable x_i. C'est à dire, à chaque étape i :

$$c(j) = \sum_{l=\max(1,i-q+j)}^{\min(n,i+j-1)} w[l]$$

En suivant la règle gloutonne présentée plus haut, $w[i]$ est instancié à 1 ssi x_i n'est pas encore instancié ($\mathcal{D}(x_i) = \{0, 1\}$) et si les contraintes de capacité ne sont pas violées , c'est à dire, qu'il n'y a pas de sous-séquence de cardinalité u ou plus impliquant x_i. Pour cela, on teste la valeur maximale de $c(j)$ pour $j \in [1, \ldots, q]$ (condition ligne 2). Dans ce cas, la cardinalité de chaque sous-séquence impliquant x_i est incrémentée (ligne 3). Enfin, lorsqu'on passe à la variable suivante, les valeurs de $c(j)$ sont décalées (ligne 4) et la valeur de $c(q)$ est obtenue à partir de sa valeur précédente en ajoutant $w[i+q]$ et en soustrayant $w[i]$ (ligne 5). Dans la suite, on note $\overrightarrow{w}$ (resp. $\overleftarrow{w}$), l'instanciation déterminée par l'algorithme `leftmost` pour la séquence $x_1.., x_n$ (resp. $x_n, .., x_1$). Pour simplifier, on note $\overleftarrow{w}[i]$ la valeur retournée par l'algorithme `leftmost` pour la variable x_i (au lieu de x_{n-i+1}).

Algorithm 1: `leftmost`

```
   count ← 0;
1  foreach i ∈ [1, ..., n] do w[i] ← min(x_i);
   foreach i ∈ [1, ..., q] do w[n + i] ← 0;
   c(1) ← w[1];
   foreach j ∈ [2, ..., q] do c(j) ← c(j − 1) + w[l];
   foreach i ∈ [1, ..., n] do
2      if |D(x_i)| > 1 & max_{j∈[1,q]}(c(j)) < u then
           w[i] ← 1;
           count ← count + 1;
3          foreach j ∈ [1, ..., q] do c(j) ← c(j) + 1;
4      foreach j ∈ [2, ..., q] do c(j − 1) ← c(j);
5      c(q) ← c(q − 1) + w[i + q] − w[i];
   return w;
```

Lemme 3.1 *`leftmost` maximise $\sum_{i=1}^{n} x_i$ sous les contraintes* ATMOSTSEQ$(u, q, [x_1, \ldots, x_n])$.

Preuve. Soit $\overrightarrow{w}$ l'instanciation déterminée par `leftmost` et supposons qu'il existe une autre instanciation w tel que $|w| > |\overrightarrow{w}|$. Soit i, le plus petit indice tel que $\overrightarrow{w}[i] \neq w[i]$. Par définition de $\overrightarrow{w}$, on sait que $\overrightarrow{w}[i] = 1$ et donc que $w[i] = 0$. Soit également j le plus petit indice tel que $\overrightarrow{w}[j] < w[j]$ (un tel indice existe car $|w| > |\overrightarrow{w}|$).

On affirme que l'instanciation w', égale à w, sauf pour $w'[i] = 1$ et $w'[j] = 0$ (comme pour $\overrightarrow{w}$) est consistante. En effet, sa cardinalité n'est pas modifiée par ce changement, donc $|w'| = |w|$. Si on considère toutes les contraintes de somme qui sont touchées par cette permutation de valeurs, c'est à dire les sommes $\sum_{l=a}^{a+q-1} w'[l]$ définies sur les intervalles $[a, a+q-1]$ tels que $a \leq i < a+q$ ou $a \leq j < a+q$. trois cas se présentent :

1. $a \leq i < j < a+q$. Dans ce cas, la valeur de la somme est identique pour w et w', donc elle est inférieure ou égale à u.
2. $i < a \leq j < a+q$. Dans ce cas, la valeur de la somme dans w' est diminuée de 1 par rapport à w, elle est donc inférieure ou égale à u.
3. $a \leq i < a+q \leq j$. Dans ce cas, pour tout $l \in [a, a+q-1]$, on a $w'[l] \leq \overrightarrow{w}[l]$ car j est le plus petit indice tel que $\overrightarrow{w}[j] < w[j]$. Donc, la somme est inférieure ou égale à u.

Donc, à partir d'une instanciation w de cardinalité maximale qui diffère de $\overrightarrow{w}$ au rang i, on peut construire une autre instanciation de même cardinalité qui ne diffère de $\overrightarrow{w}$ jusqu'à $i+1$. En itérant, on peut obtenir une instanciation identique à $\overrightarrow{w}$, mais de cardinalité $|w|$, ce qui contredit donc notre hypothèse $|w| > |\overrightarrow{w}|$. □

Corollaire 3.1 *Soit $\overrightarrow{w}$ un vecteur obtenu par `leftmost`,* ATMOSTSEQCARD$(u, q, d, [x_1, \ldots, x_n])$ *est satisfiable ssi les trois conditions suivantes sont respectées :*

$$\text{ATMOSTSEQ}(u, q, [x_1, \ldots, x_n]) \text{ est satisfiable} \quad (3.1)$$

$$\sum_{i=1}^{n} min(x_i) \leq d \quad (3.2)$$

$$|\overrightarrow{w}| \geq d \quad (3.3)$$

Preuve. Ces trois conditions sont nécessaires : (1) et (2) proviennent de la définition et (3) est une application directe du Lemme 3.1. Prouvons que ces conditions sont suffisantes en montrant que si elles sont vérifiées alors il existe une solution. Comme ATMOSTSEQ$(u, q, [x_1, \ldots, x_n])$ est satisfiable, $\overrightarrow{w}$ ne viole pas de séquence de contraintes ATMOST car la valeur 1 est affectée à x_i ssi toutes les sous-séquences impliquant x_i ont une cardinalité de $u-1$ ou moins. De plus, comme $\sum_{i=1}^{n} min(x_i) \leq d$, il y a au moins $|\overrightarrow{w}| - d$ variables telles que $min(x_i) = 0$ et $\overrightarrow{w}[i] = 1$. Instancier ces variables à 1 ne viole pas la contrainte ATMOSTSEQ. Par conséquent, il existe un support. □

Le lemme 3.1 et le corollaire 3.1 fournissent une procédure polynomiale de recherche de support pour la contrainte ATMOSTSEQCARD. En effet, la complexité dans le pire cas de l'algorithme 1 est en $O(nq)$: il comporte n étapes et pour chacune, les lignes 2, 3 et 4 nécessitent $O(q)$ opérations. Ainsi, pour chaque variable x_i, un suport pour $x_i = 0$ ou $x_i = 1$ peut être obtenu en $O(nq)$.

Donc, on a une procédure d'AC avec une complexité au pire en $O(n^2 q)$.

3.1 Filtrage des domaines

Cette partie montre que l'on peut filtrer toutes les valeurs inconsistantes pour la contrainte ATMOSTSEQCARD avec seulement deux appels à l'algorithme 1, c'est à dire avec la même complexité dans le pire cas.

Soit $w[a:b]$ la projection de l'instanciation w sur la sous-séquence $x_a, \ldots, x_b$.

En exécutant l'algorithme `leftmost` de gauche à droite et de droite à gauche sur la séquence $x_1, \ldots, x_n$, on obtient les valeurs de $|\overrightarrow{w}[1:i]|$ et de $|\overleftarrow{w}[i:n]|$ pour tout $i \in [1, \ldots, n]$. Ceci peut être exploité pour assurer l'arc-consistance de ATMOSTSEQCARD. Tout d'abord, on montre qu'il n'y a besoin de ne considérer que le cas où la cardinalité $|\overrightarrow{w}|$ du vecteur calculé par `leftmost` est exactement égale à d ; dans les autres cas la contrainte est soit valide soit insatisfiable.

Proposition 3.1 *Soit $\overrightarrow{w}$ le vecteur calculé par `leftmost`, si les trois propositions suivantes sont vérifiées :*

$$\text{ATMOSTSEQ}(u, q, [x_1, \ldots, x_n]) \text{ is AC} \tag{3.4}$$

$$\sum_{i=1}^{n} min(x_i) \leq d \tag{3.5}$$

$$|\overrightarrow{w}| > d \tag{3.6}$$

alors ATMOSTSEQCARD$(u, q, d, [x_1, \ldots, x_n])$ *est* AC.

Preuve. Avec le corollaire 3.1 on sait que ATMOSTSEQCARD$(u, q, d + 1, [x_1, \ldots, x_n])$ est satisfiable. Soit w une instanciation satisfiable, et considérons sans perte de généralité, une variable x_i telle que $|\mathcal{D}(x_i)| > 1$. Supposons tout d'abord que $w[i] = 1$. La solution w' égale à w sauf pour $w'[i] = 0$ satisfait ATMOSTSEQCARD$(u, q, d, [x_1, \ldots, x_n])$ En effet, $|w'| = |w| - 1 = d$ et comme ATMOSTSEQ$(u, q, [x_1, \ldots, x_n])$ est satisfaite par w, elle est satisfaite par w'. Ainsi, pour toute variable x_i, il existe un support pour $x_i = 0$.

Supposons que $w[i] = 0$, et soit $a < i$ (resp. $b > i$) le plus grand indice (resp. le plus petit) tel que $w[a] = 1$ et $\mathcal{D}(x_a) = \{0, 1\}$ (resp. $w[b] = 1$ et $\mathcal{D}(x_b) = \{0, 1\}$). Soit w' une instanciation telle que $w'[i] = 1$, $w'[a] = 0$, $w'[b] = 0$, et $w = w'$ d'autre part. On a $|w'| = d$, et on peut montrer qu'elle satisfait ATMOSTSEQ$(u, q, [x_1, \ldots, x_n])$. Considérons une sous-séquence $x_j, \ldots, x_{j+q-1}$. Si $j+q \leq i$ ou $j > i$ alors $\sum_{l=j}^{j+q-1} w'[l] \leq \sum_{l=j}^{j+q-1} w[l] \leq u$, ainsi seuls les indices tels que $j \leq i < j + q$ sont à prendre en compte. Il y a deux cas :

1. Soit a ou b ou les deux sont dans la sous-séquence ($j \leq a < j + q$ ou $j \leq b < j + q$). Dans ce cas, $\sum_{l=j}^{i+q-1} w'[l] \leq \sum_{l=j}^{i+q-1} w[l]$.
2. Ni a ni b ne sont dans la sous-séquence ($a < j$ et $j + q \leq b$). Dans ce cas, comme $\mathcal{D}(x_i) = \{0, 1\}$ et comme la condition 3.4 est vérifiée, on a $\sum_{l=j}^{i+q-1} \min(x_l) < u$. De plus, comme $a < j$ et $j + q \leq b$, il n'y a pas de variable x_l dans cette sous-séquence telle que $w[l] = 1$ et $0 \in \mathcal{D}(x_l)$. Ainsi, on a $\sum_{l=j}^{i+q-1} w[l] < u$, c'est à dire $\sum_{l=j}^{i+q-1} w'[l] \leq u$.

Dans les deux cas, w satisfait les contraintes de capacité. □

Rappelons que réaliser l'AC sur ATMOSTSEQ est triviale car AMONG est monotone. On se concentre donc sur le cas où ATMOSTSEQ est AC, et $|\overrightarrow{w}| = d$. Ainsi, les propositions 3.2, 3.3, 3.4 et 3.5 suivantes ne s'appliquent que dans ce cas. L'égalité $|\overrightarrow{w}| = d$ est donc implicite dans chacune d'entre elles.

Proposition 3.2 *Si $|\overrightarrow{w}[1:i-1]| + |\overleftarrow{w}[i+1:n]| < d$ alors $x_i = 0$ n'est pas* AC.

Preuve. Supposons que l'on a $|\overrightarrow{w}[1:i-1]| + |\overleftarrow{w}[i+1:n]| < d$ et qu'il existe une instanciation w' tel que $w'[i] = 0$ et $|w'| = d$.

En appliquant le Lemme 3.1 sur la séquence $x_1, \ldots, x_{i-1}$, on a $\sum_{l=1}^{i-1} w'[l] \leq |\overrightarrow{w}[1:i-1]|$.

En appliquant le Lemme 3.1 sur la séquence $x_n, \ldots, x_{i+1}$, on a $\sum_{l=i+1}^{n} w'[l] \leq |\overrightarrow{w}[i+1:n]|$.

Comme $w'[i] = 0$, on a $|w'| = \sum_{l=1}^{n} w'[l] < d$, ce qui est en condradiction avec l'hypothèse $|w'| = d$. Par conséquence, il n'y a pas de support pour $x_i = 0$. □

Proposition 3.3 *Si $|\overrightarrow{w}[1:i]| + |\overleftarrow{w}[i:n]| \leq d$ alors $x_i = 1$ n'est pas* AC.

Preuve. Supposons que l'on a $|\overrightarrow{w}[1:i]| + |\overleftarrow{w}[i:n]| \leq d$ et qu'il existe une instanciation w' tel que $w'[i] = 1$ et $|w'| = d$.

Par application du Lemme 3.1 sur la séquence $x_1, \ldots, x_i$, on a $\sum_{l=1}^{i} w'[l] \leq |\overrightarrow{w}[1:i]|$.

Par application du Lemme 3.1 sur la séquence $x_n, \ldots, x_i$, on a $\sum_{l=i}^{n} w'[l] \leq |\overleftarrow{w}[i:n]|$.

Donc, comme $w'[i] = 1$, on a $|w'| = \sum_{l=1}^{i} w'[l] + \sum_{l=i}^{n} w'[l] - 1 < d$, ce qui contredit l'hypothèse $|w'| = d$. Par conséquent, il n'y a pas de support pour $x_i = 1$. □

Les propositions 3.2 et 3.3 définissent une règle de filtrage. Dans une première passe, de la gauche vers la droite, on peut utiliser un algorithme similaire à `leftmost` pour calculer l'instanciation $|\overrightarrow{w}[1:i]|$ pour tout $i \in [1, \ldots, n]$. Dans une seconde passe, l'instanciation $|\overleftarrow{w}[i:n]|$ pour tout $i \in [1, \ldots, n]$ est déterminée en déroulant la même procédure sur la même séquence de variables mais *en sens inverse*, ie. de la droite vers la gauche. En utilisant ces instanciations, on peut utiliser alors les propositions 3.2 et 3.3 pour filtrer respectivement les valeurs 0 et 1. Cette procédure est détaillée ci-après.

On montre tout d'abord que ces deux règles sont complètes, c'est à dire si ATMOSTSEQ est AC et si la contrainte

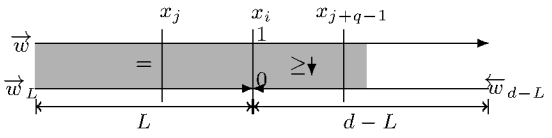


FIGURE 1: Illustration de la preuve de la Proposition 3.4. Les flêches horizontales représentent les instanciations.

de cardinalité globale est AC alors une instanciation $x_i = 0$ (resp. $x_i = 1$) est inconsistante ssi la proposition 3.2 (resp. 3.3) s'applique. Soit $\texttt{leftmost}(k)$ l'algorithme correspondant à l'application de `leftmost` s'arrêtant lorsque la cardinalité d'une instanciation atteint une valeur k donnée.

Lemme 3.2 *Soit w une instanciation satisfiable pour* ATMOSTSEQ$(u, q, [x_1, \ldots, x_n])$. *Si $k \leq |w|$ alors l'instanciation $\overrightarrow{w}_k$ déterminée par $\texttt{leftmost}(k)$ est telle que pour tout $1 \leq i \leq n$:*

$$\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq \sum_{l=i}^{n} w[l]$$

Preuve.

Il est clair que pour $i = 1$, $\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq \sum_{l=i}^{n} w[l]$. Pour $i \neq 1$, soit m l'indice pour lequel $\texttt{leftmost}(k)$ s'arrête. On distingue deux cas. Lorsque $i > m$, pour toute valeur l dans $[m+1, \ldots, n]$, on a $\overrightarrow{w}_k[l] \leq w[l]$ (car $\overrightarrow{w}_k[l] = \min(x_l)$), donc $\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq \sum_{l=i}^{n} w[l]$. Lorsque $i \leq m$, puisque $|\overrightarrow{w}_k| \leq |w|$ alors $\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq |w| - \sum_{l=1}^{i-1} \overrightarrow{w}_k[l]$. Ainsi, $\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq \sum_{l=i}^{n} w[l] + (\sum_{l=1}^{i-1} w[l] - \sum_{l=1}^{i-1} \overrightarrow{w}_k[l])$. De plus, en appliquant le lemme 3.1, la valeur $\sum_{l=1}^{i-1} \overrightarrow{w}_k[l]$ est maximale, donc supérieure ou égale à $\sum_{l=1}^{i-1} w[l]$. Par conséquent, $\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq \sum_{l=i}^{n} w[l]$. □

Proposition 3.4 *Si* ATMOSTSEQ$(u, q, [x_1, \ldots, x_n])$ *est* AC *et si $|\overrightarrow{w}[1 : i-1]| + |\overleftarrow{w}[i+1 : n]| \geq d$ alors $x_i = 0$ possède un support.*

Preuve. Soit $\overrightarrow{w}$ une instanciation déterminée par `leftmost`. On considère, sans perte de généralité, une variable x_i telle que $\mathcal{D}(x_i) = \{0, 1\}$ et $|\overrightarrow{w}[1 : i-1]| + |\overleftarrow{w}[i+1 : n]| \geq d$ et on montre que l'on peut construire un support pour $x_i = 0$. Si $\overrightarrow{w}[i] = 0$ ou $\overleftarrow{w}[i] = 0$ alors il existe un support pour $x_i = 0$, donc il n'y a besoin de ne considérer que le cas $\overrightarrow{w}[i] = \overleftarrow{w}[i] = 1$.

Soient $L = |\overrightarrow{w}[1 : i-1]|$ et $\overleftarrow{w}_{d-L}$ le résultat de $\texttt{leftmost}(d-L)$ sur la sous-séquence $x_n, \ldots, x_i$. On veut montrer que w correspondant à la concaténation de $\overrightarrow{w}[1 : i-1]$ et de $\overleftarrow{w}_{d-L}[i : n]$ est un suport pour $x_i = 0$.

Tout d'abord, on montre que $w[i] = 0$, c'est à dire $\overleftarrow{w}_{d-L}[i] = 0$. Par hypothèse, comme $|\overrightarrow{w}[1 : i-1]| + |\overleftarrow{w}[i+1 : n]| \geq d$, on a $|\overleftarrow{w}[i+1 : n]| \geq d - L$. Considérons maintenant la séquence $x_i, \ldots, x_n$, et soit w' le vecteur tel que $w'[i] = 0$ et $w' = \overleftarrow{w}[i+1 : n]$ dans les autres cas. Comme $|w'| = |\overleftarrow{w}[i+1 : n]| \geq d - L$, avec le lemme 3.2 on sait que w' a une cardinalité plus grande que $\overleftarrow{w}_{d-L}$ sur toute sous-séquence débutant à i, donc $w[i] = \overleftarrow{w}_{d-L}[i] = w'[i] = 0$.

On montre maintenant que w ne viole pas la contrainte ATMOSTSEQCARD. De manière évidente, comme cette instanciation est la concaténation de deux instanciations consistantes, elle ne peut violer la contrainte qu'à la jonction, i.e. sur une sous-séquence $x_j, \ldots, x_{j+q-1}$ telle que $j \leq i$ et $i < j + q$.

On montre que la somme de toute sous-séquence est inférieure ou égale à u en les comparant avec $\overrightarrow{w}$, comme illustré dans la Figure 1. On a $\sum_{l=j}^{j+q-1} \overrightarrow{w}[l] \leq u$ et $\sum_{l=j}^{i-1} \overrightarrow{w}[l] = \sum_{l=j}^{i-1} w[l]$. De plus, avec le lemme 3.2 comme $|\overrightarrow{w}[i : n]| = |\overleftarrow{w}_{d-L}| = d - L$, on a $\sum_{l=i}^{j+q-1} \overleftarrow{w}_{d-L}[l] \leq \sum_{l=i}^{j+q-1} \overrightarrow{w}[l]$, donc $\sum_{l=i}^{j+q-1} w[l] \leq \sum_{l=i}^{j+q-1} \overrightarrow{w}[l]$. Par conséquent, on conclue que $\sum_{l=j}^{j+q-1} w[l] \leq u$. □

Proposition 3.5 *Si* ATMOSTSEQ$(u, q, [x_1, \ldots, x_n])$ *est* AC *et si $|w[1 : i]| + |w[n : i]| > d$ alors il y a un support pour $x_i = 1$.*

Preuve. Soient $\overrightarrow{w}$ et $\overleftarrow{w}$ deux instanciations déterminées par `leftmost` sur respectivement $x_1, \ldots, x_n$ et $x_n, \ldots, x_1$. On considère sans perte de généralité une variable x_i telle que $\mathcal{D}(x_i) = \{0, 1\}$ et $|\overrightarrow{w}[1 : i]| + |\overleftarrow{w}[i : n]| > d$ et on montre que l'on peut construire un support pour $x_i = 1$. Si $\overrightarrow{w}[i] = 1$ ou $\overleftarrow{w}[i] = 1$ alors il existe un support pour $x_i = 1$, donc il n'y a besoin de ne considérer que le cas $\overrightarrow{w}[i] = \overleftarrow{w}[i] = 0$.

Soit $L = |\overrightarrow{w}[1 : i]| = |\overrightarrow{w}[1 : i-1]|$ (cette égalité est vérifiée car $\overrightarrow{w}[i] = 0$). Soit $\overrightarrow{w}_{L-1}$ une instanciation obtenue en utilisant $\texttt{leftmost}(L-1)$ sur la sous-séquence $x_1, \ldots, x_{i-1}$ et soit $\overleftarrow{w}_{d-L}$ une instanciation obtenue par $\texttt{leftmost}(d-L)$ sur la sous-séquence $x_n, \ldots, x_{i+1}$.

On montre que w vallant $w[i] = 1$, $\overrightarrow{w}_{L-1}$ sur $x_1, \ldots, x_{i-1}$ et $\overleftarrow{w}_{d-L}$ sur $x_{i+1}, \ldots, x_n$, est un support.

De manière évidente, $|w| = d$, ainsi il reste à vérifier que seules les contraintes de capacité sont satisfaites. De plus, comme cette instanciation est la concaténation de deux instanciations consistantes, elle peut violer la contrainte qu'à la jonction, i.e., pour une sous-séquence $x_j, \ldots, x_{j+q-1}$ telle que $j \leq i$ et $i < j + q$.

On montre que la somme de toute sous-séquence est inférieure ou égale à u en la comparant avec $\overrightarrow{w}$ et

$\overleftarrow{w}_{d-L}$. Tout d'abord, notons que sur la sous-séquence $x_1, \ldots, x_{i-1}$, $\overrightarrow{w}_{L-1} = \overrightarrow{w}$, sauf pour le plus grand indice a tel que $\overrightarrow{w}[a] = 1$ et $w[a] = 0$. De la même façon, sur $x_n, \ldots, x_{i+1}$, on a $\overleftarrow{w}_{d-L} = \overleftarrow{w}_{d-L+1}$, sauf pour le plus petit indice b tel que $\overleftarrow{w}_{d-L+1}[b] = 1$. On distingue deux cas. Si $j > a$, alors $\sum_{l=j}^{j+q-1} w[l] = \sum_{l=i}^{j+q-1} \overleftarrow{w}_{d-L+1}[l]$ si $j + q - 1 \geq b$, et vaut 1 autrement. Et donc inférieur ou égal à u car $i \geq j$ et $u \geq 1$. Si $j \leq a$. Dans ce cas, considérons la sous-séquence $x_j, \ldots, x_i$. Sur cet intervalle, la cardinalité de w est la même que celle de $\overrightarrow{w}$, i.e., $\sum_{l=j}^{i} w[l] = \sum_{l=j}^{i-1} \overrightarrow{w}_{L-1}[l] + 1 = \sum_{l=j}^{i} \overrightarrow{w}[l]$. Sur la sous-séquence $x_{i+1}, \ldots, x_{j+q-1}$, notons que $|w[i+1 : n]| = |\overrightarrow{w}[i+1 : n]| = d - L$, donc par le lemme 3.2, on a $\sum_{l=i+1}^{j+q-1} w[l] = \sum_{l=i+1}^{j+q-1} \overleftarrow{w}_{d-L}[l] \leq \sum_{l=i+1}^{j+q-1} \overrightarrow{w}[l]$. Par conséquent, $\sum_{l=j}^{j+q-1} w[l] \leq \sum_{l=j}^{j+q-1} \overrightarrow{w}[l] \leq u$. □

3.2 Complexité

Avec les propositions 3.2, 3.3, 3.4 et 3.5 on peut concevoir un algorithme de filtrage avec la même complexité au pire que `leftmost`. Dans cette section, une impleméntation en temps linéaire de `leftmost` est proposée, appelée `leftmost_count`, qui retourne les valeurs de $\overrightarrow{w}[1 : i]$ pour tout i (Algorithme 2).

Algorithm 2: `leftmost_count`

```
Data: [x_1, ..., x_n], u, q, d,
Result: count : [0, ..., n] ↦ [0, ..., n − 1]
foreach i ∈ [1, ..., n] do
    w[i] ← min(x_i);
    occ[i] = 0;
foreach i ∈ [0, ..., n] do count[i] ← 0;
c[0] ← w[1];
foreach i ∈ [1, ..., u + 1] do occ[n + i] = 0;
foreach i ∈ [1, ..., q] do
    w[n + i] ← 0;
    c[i] ← c[i − 1] + w[i + 1];
    occ[n + c[i − 1]] ← occ[n + c[i − 1]] + 1;
max_c ← max({c[i] | i ∈ [1, ..., q]});
foreach i ∈ [1, ..., n] do
1       if max_c < u & |D(x_i)| > 1 then
            max_c ← max_c + 1;
            count[i] ← count[i − 1] + 1;
            w[i] ← 1;
        else
            count[i] ← count[i − 1];
        prev ← c[i − 1 mod q];
        next ← c[i + q − 2 mod q] + w[i + q] − w[i];
        c[i − 1 mod q] ← next;
        if prev ≠ next then
            occ[n + next] ← occ[n + next] + 1;
            if next + count > max_c then
                max_c ← max_c + 1;
            occ[n + prev] ← occ[n + prev] − 1;
            if occ[n + prev] = 0 & prev + count = max_c then
                max_c ← max_c − 1;
return count;
```

Il est simple de voir que `leftmost_count` a une complexité au pire en $O(n)$. Pour prouver sa validité, on montre qu'une instanciation déterminée par `leftmost_count` est identique à celle déterminée par `leftmost`.

Preuve. On donne ici l'idée générale de la preuve. Les trois invariants ci-dessous sont vrais à chaque étape i de la boucle principale :

- la cardinalité de la j^{ime} sous-séquence est donnée par $c[i + j - 2 \bmod q] + count[i - 1]$.
- le nombre de sous-séquences de cardinalité k est donné par $occ[n - count[i - 1] + k]$.
- la cardinalité maximale de toute sous-séquence est donnée par max_c.

1^{ier} invariant : La valeur de $c[j]$ est actualisée de deux façons dans `leftmost`. Tout d'abord, à chaque étape de la boucle, les valeurs de $c[1]$ à $c[q]$ sont décalées vers la gauche. Donc, il n'y a qu'une seule valeur nouvelle. En utilisant l'opération modulo, on peut mettre à jour seulement ces valeurs. Puis, lorsque $w[i]$ est fixé à 1, on incrémente $c[1]$ à $c[q]$. Comme cela se produit exactement $count[i - 1]$ au début de l'étape i, on peut simplement ajouter $count[i - 1]$ pour obtenir la même valeur que dans `leftmost`.

2^{ime} invariant : La structure de données occ est un tableau mémorisant, pour l'instanciation courante w, à l'indice $n + k$, le nombre de sous-séquence incluant x_i avec une cardinalité k Donc, en décrémentant le pointeur vers le premier élément du tableau, on décale en pratique le tableau complet. Ici aussi, la valeur de $count[i-1]$, ou plutôt l'expression $(n - count[i - 1])$, pointe exactement sur le début recherché dans le tableau.

3^{ime} invariant : La cardinalité maximale (ou minimale) des sous-séquences incluant x_i, peut changer au plus d'une unité au cours de chaque étape. Donc, quand la variable max_c doit changer, elle peut être juste incrémentée (si $occ[n - count[i - 1] + max\_c + 1]$ passe de 0 à 1) ou décrémentée (si $occ[n - count[i - 1] + max\_c]$ passe de 1 à 0). □

Algorithm 3: AC(ATMOSTSEQCARD)

```
Data: [x_1, ..., x_n], u, q, d
Result: AC on AtMostSeqCard(u, q, d, [x_1, .., x_n])
AC(AtMostSeq)(u, q, [x_1, ..., x_n]);
ub ← d − Σ_{i=1}^{n} min(x_i);
L ← leftmost_count([x_1, ..., x_n], u, q, d);
if L[n] = ub then
    R ← leftmost_count([x_n, ..., x_1], u, q, d);
    foreach i ∈ [1, ..., n] such that D(x_i) = {0, 1} do
        if L[i] + R[n − i + 1] ≤ ub then D(x_i) ← {0};
        if L[i − 1] + R[n − i] ≤ ub then D(x_i) ← {1};
```

L'algorithme 3 détermine l'AC de la contrainte ATMOSTSEQCARD$(u, q, d, [x_1, \ldots, x_n])$. A la première ligne, l'AC de la contrainte ATMOSTSEQ$(u, q, [x_1, \ldots, x_n])$ est établie. Cela permet d'assurer que la règle de filtrage exposée dans ce papier est vérifiée. Pour des raisons de place, on ne donne pas le pseudo-code pour l'AC de ATMOSTSEQ mais elle peut être réalisée en temps linéaire en utilisant une procédure

$\mathcal{D}(x_i)$		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1	
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1	
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1	
$L[i]$	0	1	1	2	3	4	4	4	4	4	4	4	5	6	7	7	7	7	8	8	9	10	10	
$R[n-i+1]$		10	9	9	9	8	7	6	6	6	6	6	6	5	4	3	3	3	3	3	2	1	0	0
$L[i]+R[n-i+1]$		11	10	11	12	12	11	**10**	**10**	10	10	10	11	11	11	**10**	**10**	**10**	11	11	11	11	10	
$L[i-1]+R[n-i]$		**9**	10	10	10	10	10	10	10	10	10	10	**9**	**9**	**9**	10	10	10	10	10	**9**	**9**	10	
AC($\mathcal{D}(x_i)$)		**1**	0	.	.	.	.	**0**	**0**	0	1	0	**1**	**1**	**1**	**0**	**0**	**0**	.	.	**1**		1	

FIGURE 2: Exemple de déroulement de l'algorithme 3 pour $u = 4$, $q = 8$, $d = 12$. La première ligne représente les domaines courrants, les points représentent des variables non instanciées (donc $ub = 10$). Les deux lignes suivantes donnent les instanciations $\overrightarrow{w}$ et $\overleftarrow{w}$ obtenues par `leftmost_count` de gauche à droite et de droite à gauche. Les troisième et quatrième lignes donnent les valeurs de $L[i] = |\overrightarrow{w}[1:i]|$ ou $R[n-i+1] = |\overleftarrow{w}[i:n]|$. Les cinquième et sixième ligne correspondent à l'application des propositions 3.2 et 3.3 respectvement. La septième ligne donne l'arc-consistance des domaines

similaire à `leftmost_count`. On cherche à calculer, pour une instanciation correspondant à la borne inférieure de chaque domaine, si une variable non instanciée est couverte par une sous-séquence de cardinalité u pour les bornes inférieures des domaines. Ceci est effectué avec une version tronquée de `leftmost_count` : les valeur de $w[i]$ ne sont jamais mises à jour, i.e., elles sont fixées à leur valeur minimale et on n'entre pas dans le bloc « if-then-else » (condition 1 de l'Algorithme 2). En plus, on mémorise dans un tableau la valeur de max_c pour tout i. Ce tableau est parcouru pour effecter la valeur 0 à chaque variable non instanciée couverte par au moins une sous-séquence de cardinalité u afin de réaliser l'AC de ATMOSTSEQ.

La suite est une application directe des propositions 3.2, 3.3, 3.4 et 3.5. Un exemple d'exécution est donné dans la Figure 2. La complexité au pire de l'Algorithme 3 est ainsi en $O(n)$, donc optimale.

4 Résultats expérimentaux

L'algorithme de propagation a été testé sur des instances de car-sequencing et de crew-rostering. Les expérimentations ont été effectuées sur un processeur Intel Xeon à 2.67GHz sous Linux. Les développements ont été faits avec Ilog-Solver. 5 exécutions aléatoires de chaque instance ont été lancées avec un temps limite de 20 minutes. [3]

Pour comparer l'efficacité relatives des propagations, les résultats sont moyennés sur plusieurs heuristiques pour réduire le biais que celles-ci pourraient introduire. Pour chaque ensemble de données, *#sol* représente le nombre d'instances résolues. Puis, on donne le temps CPU (*time*) en secondes et le nombre de retour-arrière (*avg bts*) (tous les deux moyennés sur le nombre de solutions obtenues, le nombre d'instances et d'heuristiques) ainsi que le nombre maximum de retour-arrière (*max bts*). Les résultats (en termes de *#sol*) de la meilleure méthode sont notés en gras.

3. pour un temps total de CPU d'environ 200 jours.

4.1 Car-sequencing

Pour le car-sequencing [8, 17], n véhicules doivent être fabriqués sur une ligne d'assemblage tout en respectant des contraintes de capacités et de demande.

Nous nous sommes basés sur le modèle standard implémenté avec Ilog-Solver. Il y a n variables entières représentant les classes de véhicules de chaque position dans la ligne d'assemblage et nm variables booléennes y_i^j représentant le fait que le véhicule placé en i^{ime} position nécessite l'option j. La demande de chaque classe est exprimée avec une contrainte de cardinalité globale (GCC) et l'arc-consistance de cette contrainte est réalisée avec Ilog [14]. Quatre modélisations pour les contraintes de capacité sont comparées (disant que pour chaque option j, chaque sous-séquence de taille q_j contient au plus u_j variables d'option fixées à 1) associées aux contraintes de demande de chaque option (déduites des contraintes de demande sur chaque classe). La première modélisation (*sum*) comprend une simple décomposition en une séquence de contraintes somme pour les contraintes de capacité ainsi qu'une contrainte somme supplémentaire exprimant la demande. La seconde modélisation, (*gsc*) utilise une contrainte GSC par option. La troisième, (*amsc*) est une application de la procédure d'AC introduite dans cet article pour la contrainte ATMOSTSEQCARD. Enfin, la quatrième modélisation, (*amsc+gsc*) combine les contraintes ATMOSTSEQCARDet GSC.

34 heuristiques obtenues par combinaisons de différents paramètres ont été utilisées : *exploration* de la ligne d'assemblage soit de manière lexicographique soit depuis le milieu vers les bords ; *branchement* sur l'affectation d'une classe ou d'une option à une position dans la ligne ; *sélection* de la meilleure classe ou option à l'aide de différents critères évidents (demande maximum, ratio u/q minimum) ou de critères plus complexes dérivés de ceux proposés dans [5, 16].

3 groupes d'instances de la CSPLib [9] ont été considérés. Le premier groupe, appelé set1 par la suite, comporte 70 instances à 200 véhicules, toutes satisfiables. Dans le

second groupe, il y a 9 instances de 100 véhicules réparties en 4 satisfiables, notées set2 et 5 instances insatisfiables, noté set3. Le troisième groupe contient 30 instances de plus grande taille (jusqu'à 400 véhicules), parmi elles, on a considéré les 7 instances connues pour être satisfiables, notées set4.

Les résultats sont présentés dans la Table 1. Dans tous les cas, la meilleure valeur en terme de nombre de problèmes résolus est obtenue soit avec *amsc+gsc* (pour les instances de petite taille ou insatisfiables) ou avec *amsc* seul (pour les instances de plus grande taille set2 et set4). Globalement, on note que GSC permet d'élaguer beaucoup plus de valeurs incohérentes que ATMOSTSEQCARD. Toutefois, la propagation de GSC ralentit très significativement la recherche (on a observé un facteur 12.5 sur le nombre de nœuds explorés par seconde). Par ailleurs, les niveaux de filtrage obtenus par ces deux méthodes sont incomparables. En conséquent combiner ces deux contraintes est toujours bénéfique plutôt que d'utiliser GSC seule.

Dans [18], les auteurs ont évalué leur proposition sur les instances set1, set2 et set3. Dans leurs expérimentations, ils ont considéré les meilleurs résultats obtenus pour 2 heuristiques (σ et min domain). En propageant uniquement COST-REGULAR ou GEN-SEQUENCE 50.7% des instances sont résolues alors qu'en les combinant avec GSC 65.2% sont résolues (avec un temps maximum de 1h). Dans nos expérimentations, en moyenne sur les 34 heuristiques et sur les 5 exécutions aléatoires, ATMOSTSEQCARD et GSC résolvent respectivement 82.5% et 86.11% des instances et la combinaison des deux permet de résoudre 86.36% des instances avec un temps CPU limite de 20 minutes.

4.2 Crew-rostering

Pour le crew-rostering, des périodes de travail doivent être allouées à des employés sur un horizon donné de telle sorte qu'il y ait suffisament de personnes pour assurer un service à chaque période et que les contraintes sur l'organisation du travail soit respectées. Cette dernière condition peut entraîner une grande variété de contraintes. Des travaux antérieurs [11, 13] s'appuient sur des motifs autorisés (ou interdits) pour exprimer des contraintes sur des périodes successives. Par exemple, s'il y a 3 périodes de 8 heures par jour : J (jour), S (soir) et N (nuit), NJ peut être interdit pour exprimer le fait que les employés ont besoin d'un repos après une période de travail de nuit. Cet article considère un cas simple avec 20 employés, 3 périodes de travail de 8 heures par jour sur lesquelles on ne peut travailler plus de 8 heures par jour, pas plus de 5 jours par semaines et que les pauses entre deux périodes de travail doivent être d'au minimum 16 heures. L'horizon est de 28 jours, chaque employé doit travailler en moyenne 34 heures par semaine (17 périodes sur 4 semaines).

La modélisation se base sur une variable booléenne e_{ij} pour chacun des m employés et des n périodes traduisant le fait qu'un employé i travaille en période j. La demande d_j^s de chaque période j est propagée avec une contrainte de somme : $\sum_{i=1}^{m} e_{ij} = d_j^s$.[4] Les autres contraintes sont posées en utilisant deux contraintes ATMOSTSEQCARD par employé : l'une avec le ratio $u/q = 1/3$, l'autre avec le ratio $5/21$, les deux contraintes ont la même demande $d = 17$ correspondant à 34 heures de travail par semaine. Trois modélisations sont comparées. Dans la première, (*sum*), ces deux contraintes sont décomposées avec une séquence de contraintes de somme. La seconde, (*gsc*) utilise une contrainte GSC. Notons que dans ce cas, comme les variables sont booléenne, la contrainte GCC de GSC n'est rien d'autre qu'une contrainte de somme. Donc, elle ne peut pas élaguer plus que la contrainte ATMOSTSEQCARD (mais elle est plus puissante que la décomposition en somme). La troisième modélisation, (*amsc*) emploie la contrainte ATMOSTSEQCARD présentée dans cet article.

Les demandes sur l'horizon sont régulières sur les 4 semaines : pour les périodes de jour et de soir, 6 employés sont nécessaires en semaine et 2 pour le week-end ; sur les périodes de nuit 3 employés sont nécessaires en semaine et seulement 1 en week end. De plus, un ensemble d'indisponibilités a été générées aléatoirement pour chaque employé sur les différentes période. 281 instances ont été générées, avec des indisponibilités de 18% à 46% par pas de 0.1. Ces instances peuvent être partitionnées en trois groupes : le premier groupe comporte 126 instances avec peu d'indisponibilités (toutes les instances sont satisfiables), le troisième groupe est composé de 44 instances (principalement insatisfiables) avec un fort ratio d'indisponibilité, le reste des instances constituent le second groupe.

L'heuristique de branchement utilisée est la suivante : pour chaque employé i on sélectionne celui ayant un slack minimal $\sigma_i = n_i - \frac{21 d_i}{5}$ et on l'affecte à une période possible ayant le slack minimum $\sigma'_j = m_j - d_j^s$, où n_i (resp. m_j) représente le nombre de variables e_{ij} (resp. $e_{kj}, k \in \{1..20\}$) non instanciées et d_i (resp. d_j^s) le nombre résiduel de périodes devant être travaillées par l'employé i (resp. d'employés devant être affectés à une période j). De plus, on utilise une seconde heuristique s'appuyant sur les mêmes principes mais sélectionnant d'abord la période puis l'employé.

Les résultats obtenus sont données dans la Table 2. L'algorithme d'AC proposé réalise plus de filtrage que la décomposition en contraintes somme et que la contrainte GSC. Cependant, selon les heuristiques et les exécutions aléatoires, la taille de l'arbre de recherche n'est pas toujours la plus petite. On observe que notre filtrage ATMOSTSEQCARD n'est que marginalement plus lent en

4. Les instances générées ont exactement autant d'employés que nécessaire pour assurer le service total, on peut ainsi utiliser une contrainte d'égalité.

TABLE 1: Evaluation des méthodes de filtrage (en moyenne sur toutes les heuristiques)

Méthodes	set1 ($70 \times 34 \times 5$)				set2 ($4 \times 34 \times 5$)				set3 ($5 \times 34 \times 5$)				set4 ($7 \times 34 \times 5$)			
	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time
sum	8480	231.2K	25.0M	13.93	95	1.4M	15.3M	76.60	0	-	-	> 1200	64	543.3K	13.7M	43.81
gsc	11218	1.7K	2.3M	3.60	325	131.7K	1.5M	110.99	31	55.3K	218.5K	276.06	140	25.2K	321.9K	56.61
amsc	10702	39.1K	13.8M	4.43	**360**	**690.8K**	**10.2M**	**72.00**	16	40.3K	83.4K	8.62	**153**	**201.4K**	**3.2M**	**33.56**
amsc+gsc	**11243**	**1.2K**	**1.1M**	**3.43**	339	118.4K	1.0M	106.53	**32**	**57.7K**	**244.7K**	**285.43**	147	23.8K	371.0K	66.45

TABLE 2: Evaluation des méthodes de filtrage (en moyenne sur toutes les heuristiques)

Benchmarks	sous-contraints ($5 \times 2 \times 126$)				difficiles ($5 \times 2 \times 111$)				sur-contraints ($5 \times 2 \times 44$)			
	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time
sum	1229	110.5K	10.1M	12.72	574	370.7K	13.4M	38.45	272	52.1K	5.7M	5.56
gsc	1210	6.2K	297.2K	29.19	579	23.5K	433.5K	77.78	276	7.7K	378.9	24.14
amsc	**1237**	**34.2K**	**7.5M**	**5.82**	**670**	**213.4K**	**7.5M**	**31.01**	**284**	**51.3**	**7.5M**	**6.22**

termes de nœuds explorés par seconde que la décomposition en somme et beaucoup plus rapide (avec un facteur 20.4) que GSC. Il peut résoudre 15.7% et 16.7% de problèmes en plus (dans le temps limite de 20 minutes) parmi les instance les plus difficiles que la contrainte GSC et la décomposition en somme, respectivement.

5 Conclusion

Cet article propose un algorithme linéaire pour réaliser l'arc-consistance de la contrainte ATMOSTSEQCARD. Les évaluations expérimentales, sur des problèmes de car-sequencing et de crew-rostering, ont montré que cette contrainte pouvait être utile pour ces applications.

Remerciements : Les auteurs tiennent à remercier Nina Narodytska pour ses commentaires constructifs.

Références

[1] N. Beldiceanu and M. Carlsson. Revisiting the Cardinality Operator and Introducing the Cardinality-Path Constraint Family. In *ICLP*, pages 59–73, 2001.

[2] N. Beldiceanu and E. Contejean. Introducing Global Constraints in CHIP. *Mathematical Computation Modelling*, 20(12) :97–123, 1994.

[3] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. The Slide Meta-Constraint. In *CPAI Workshop, held alongside CP*, 2006.

[4] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Slide : A Useful Special Case of the Card-path Constraint. In *ECAI*, pages 475–479, 2008.

[5] S. Boivin, M. Gravel, M. Krajecki, and C. Gagné. Résolution du Problème de Car-sequencing à l'Aide d'une Approche de Type FC. In *JFPC*, 2005.

[6] S. Brand, N. Narodytska, C.-G. Quimper, P. J. Stuckey, and T. Walsh. Encodings of the Sequence Constraint. In *CP*, pages 210–224, 2007.

[7] S. Demassey, G. Pesant, and L.-M. Rousseau. A Cost-Regular Based Hybrid Column Generation Approach. *Constraints*, 11(4) :315–333, 2006.

[8] M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving the Car-Sequencing Problem in Constraint Logic Programming. In *ECAI*, pages 290–295, 1988.

[9] I. P. Gent and T. Walsh. Csplib : a benchmark library for constraints, 1999.

[10] M. J. Maher, N. Narodytska, C.-G. Quimper, and T. Walsh. Flow-Based Propagators for the SEQUENCE and Related Global Constraints. In *CP*, pages 159–174, 2008.

[11] J. Menana and S. Demassey. Sequencing and Counting with the multicost-regular Constraint. In *CPAIOR*, pages 178–192, 2009.

[12] G. Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. In *CP*, pages 482–495, 2004.

[13] G. Pesant. Constraint-Based Rostering. In *PATAT*, 2008.

[14] J.-C. Régin. Generalized Arc Consistency for Global Cardinality Constraint. In *AAAI*, pages 209–215(2), 1996.

[15] J.-C. Régin and J.-F. Puget. A Filtering Algorithm for Global Sequencing Constraints. In *CP*, pages 32–46, 1997.

[16] B.M. Smith. Succeed-first or Fail-first : A Case Study in Variable and Value Ordering, 1996.

[17] C. Solnon, V. Cung, A. Nguyen, and C. Artigues. The car sequencing problem : Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *EJOR*, 191 :912–927, 2008.

[18] W. J. van Hoeve, G. Pesant, L.-M.Rousseau, and A. Sabharwal. New Filtering Algorithms for Combinations of Among Constraints. *Constraints*, 14(2) :273–292, June 2009.

[19] W. J. van Hoeve, G. Pesant, L.-M. Rousseau, and Ashish Sabharwal. Revisiting the Sequence Constraint. In *CP*, pages 620–634, 2006.

Résolution Etendue par Substitution Dynamique des Fonctions Booléennes

Said Jabbour[1] **Jerry Lonlac**[1,2] **Lakhdar Saïs**[1]

[1] CRIL - CNRS - Université Lille-Nord de France F-62307 Lens Cedex
[2] Département d'Informatique - Université de Yaoundé 1 B.P. 812 Yaoundé, Cameroun
{jabbour,lonlac,sais}@cril.fr

Abstract

Ce travail présente une technique originale de substitution dynamique des fonctions booléennes. Il détecte premièrement un ensemble de fonctions booléennes dans la formule booléenne sous forme normale conjonctive (CNF). Ces fonctions sont ensuite utilisées pour réduire la taille des différentes clauses apprises en substituant les arguments d'entrée par les arguments de sortie. Ceci conduit à une façon originale d'intégrer une forme restreinte de résolution étendue, l'un des plus puissants systèmes de preuve par résolution. En effet, la plupart des fonctions booléennes extraites correspondent à celles introduites au cours de la phase d'encodage CNF en appliquant le principe d'extension Tseitin. Substituer les entrées par les sorties peut être vu comme une façon élégante d'imiter la résolution étendue avec la manipulation des sous-formules. Des expérimentations préliminaires montrent la faisabilité de notre approche sur certaines classes d'instances SAT prises des récentes compétitions SAT et SAT-Race.

1 Introduction

Le problème SAT, i.e., la vérification de la fatisfaction d'un ensemble de clauses, est central dans plusieurs domaines et notamment en intelligence artificielle incluant le problème de satisfaction de contraintes (CSP), planification, raisonnement non-monotone, vérification de logiciels, etc. Aujourd'hui, SAT a gagné une audience considérable avec l'apparition d'une nouvelle génération de solveurs SAT capable de résoudre de très grandes instances issues du codage des applications du monde réel ainsi que par le fait que ces solveurs constituent d'importants composants de base pour plusieurs domaines, e.g., SMT (SAT modulo théorie), preuve de théorème, comptage de modèles, problème QBF, etc. Ces solveurs appelés solveurs SAT modernes [23, 13], sont basés sur la propagation de contraintes classiques [11] éfficacement combinée à d'éfficaces structures de données avec : (i) stratégies de redémarrage [16, 19], (ii) activité basée sur l'heuristique de choix de variables (comme VSIDS) [23], et (iii) apprentissage de clauses [22, 6, 23]. l'apprentissage de clauses est maintenant connu comme l'un des éléments les plus importants des solveurs SAT modernes. L'idée principale est que quand une branche courante de l'arbre de recherche conduit à un conflit, l'apprentissage de clauses vise à dériver une clause qui succinctement exprime les causes du conflit. Cette clause apprise est ensuite utilisée pour élaguer l'espace de recherche. L'apprentissage de clauses également connu dans la littérature comme "Conflict Driven Clause Learning" (CDCL) se réfère maintenant au plus connu et utilisé schéma d'apprentissage premier UIP, d'abord intégré dans les solveurs SAT Grasp [22] et éfficacement mise en œuvre dans zChaff [23]. Les solveurs SAT modernes peuvent être vus comme une extension de la bien connue procédure DPLL classique obtenue grâce à différentes améliorations. Il est important de noter que la bien connue règle de résolution joue encore un rôle prépondérant dans l'efficacité des solveurs SAT modernes qui peut être vu comme une forme particulière de la résolution générale [7].

Théoriquement, en intégrant l'apprentissage de clause à la procédure DPLL classique [11], le solveur SAT obtenu formulé comme un système de preuve est aussi puissant que la résolution générale [25]. Ce résultat théorique important, suggère que, pour améliorer l'efficacité des solveurs SAT, on a besoin de trouver des systèmes de preuves plus puissants. Cette question de recherche est également motivée par l'existence de plus en plus des applications challengeant les ins-

tances SAT qui ne peuvent actuellement être résolues par les solveurs SAT disponibles. Le résolution étendue [27] et la résolution symétrie [21] sont les deux systèmes de preuve qui sont connus comme étant plus puissant que la résolution. Le principe d'extension permet l'introduction des variables auxiliaires pour représenter les formules intermédiaires de telle sorte que la longueur d'une preuve peut être considérablement réduite en manipulant ces variables au lieu de manipuler les formules qu'elles représentent. La symétrie, d'autre part, permet de reconnaître qu'une formule reste invariante sous certaines permutations de noms de variables, et utilise cette information pour éviter de répéter les dérivations indépendantes des formules intermédiaires qui ne sont que des variantes permutationnelles d'une autre formule.

Pour la symétrie, plusieurs approches ont été proposées à la fois dans la satisfiabilité booléenne (eg. [10, 8, 1]) et dans la programmation par contraintes (e.g. [26, 15, 14]). Cependant, l'automatisation du système de preuve par résolution étendue reste encore une question ouverte. La résolution étendue ajoute une règle d'extension au système de preuve par résolution, permettant l'introduction de définitions dans la preuve. Par exemple étant donnée une formule booléenne $\mathcal{F}$, contenant les variables a et b, et v une nouvelle variable, on peut ajouter la définiton $v \leftrightarrow a \vee b$ tout en préservant la satisfiabilité. Rappelons que la règle d'extension est à la base de la transformation linéaire des formules booléennes générales en forme normale conjonctive (CNF). Par exemple, pour $n > 4$, la formule $\mathcal{F} = (x_1 \leftrightarrow x_2 \leftrightarrow \cdots \leftrightarrow x_n)$ n'a aucune formule polynomialement équivalente dans la représentation $CNF(\mathcal{F})$. Cependant, en utilisant le principe d'extension, on peut obtenir une transformation linéaire de $\mathcal{F}$ à $CNF(\mathcal{F})$. La formule CNF obtenue est équivalente à celle d'origine par rapport à la satisfiabilité. Il est également démontré que certaines formules qui sont difficiles pour la résolution (e.g. pigeon-hole formulas [18]) admettent des courtes preuves par résolution étendue [9]. La courte preuve pour le problème des pigeons est obtenue par simulation à l'aide de l'induction de la règle d'extension. Comme mentionné par B. Krishnamurthy dans [20], l'extension permet la définition d'hypothèses intermédiaires qui peuvent être introduites par la définition de nouvelles variables reposant sur ces hypothèses. En manipulant ces hypothèses, de courtes preuves peuvent être obtenues pour certaines formules difficiles.

Récemment, un solveur SAT CDCL basé sur une restriction de l'application de la règle d'extension a été proposé dans [3]. Plus précisement, quand 2 clauses successives apprises par le solveur sont de la forme $l_1 \vee \alpha$, $l_2 \vee \alpha$, une nouvelle variable $z \leftrightarrow \neg l_1 \vee \neg l_2)$ est introduite.

La principale difficulté derrière l'automatisation de la résolution étendue est de déterminer (1) quand est ce que de telles définitions seront ajoutées et (2) quelles fonctions booléennes représenteront elles. L'approche proposée dans ce papier exploite les nouvelles variables introduites dans la phase d'encodage des formules booléennes générales pour CNF grâce à l'application du principe d'extension de Tseitin. Substituer ces variables pendant la recherche peut être vu comme une façon élégante de manipuler dynamiquement ces sous-formules. Cela imite clairement le principe de la résolution étendue. Notre proposition répond à la fois aux questions (1) et (2), en exploitant les fonctions booléennes cachées généralement introduites lors de l'encodage, et en les utilisant afin de minimiser les clauses apprises. De plus, comme on peut le voir plus tard, notre approche proposée à une certaine similitude et différence avec le raisonnement binaire et la règle d'hyper résolution [5, 4].

Le papier est organisé comme suit. Après quelques notations et définitions préliminaires, quelques notions théoriques autours des solveurs SAT, fonctions booléennes (section 2), notre substitution dynamique des fonctions booléennes est décrite dans la section 3. Finalement, avant de conclure, les résultats expérimentaux démontrant la faisabilité de notre approche sont présentés.

2 Définitions

2.1 Définitions et notations préliminaires

Une *formule CNF* $\mathcal{F}$ est une conjonction de *clauses*, où une clause est une disjonction de *littéraux*. Un littéral est interprété comme une variable propositionnelle positive (x) ou negative ($\neg x$). Les deux littéraux x et $\neg x$ sont appélés *complémentaire*. On note par $\bar{l}$ le littéral complémentaire de l. Pour un ensemble de littéraux L, $\bar{L}$ est défini comme $\{\bar{l} \mid l \in L\}$. Une clause *unitaire* est une clause contenant seulement un seul littéral (appelé *littéral unitaire*), tandis qu'une clause binaire contient excatement deux littéraux. Une *clause vide* , notée $\bot$, est interprétée comme fausse (insatisfiable), alors qu'une *formule CNF vide*, notée $\top$, est interprétée comme vraie (satisfiable).

L'ensemble des variables apparaissant dans $\mathcal{F}$ est noté $V_{\mathcal{F}}$. Un ensemble de littéraux est *complet* s'il contient un littéral pour chaque variable de $V_{\mathcal{F}}$, et *fondamental* s'il ne contient pas de littéraux complémentaires. Une *affectation* ρ d'une formule booléenne $\mathcal{F}$ est une fonction qui associe la valeur $\rho(x) \in \{false, true\}$ à quelques variables de $x \in \mathcal{F}$. ρ est *complète* s'il attribue une valeur pour chaque

variable $x \in \mathcal{F}$, et *partielle* sinon. Une affectation est alternativement représentée par un ensemble de littéraux complet et fondamental, de façon évidente un *modèle* d'une formule $\mathcal{F}$ est une affectation ρ qui laisse la formule *vraie* ; notée $\rho \models \Sigma$.

On dénote par $\eta[x, c_i, c_j]$ la *résolvante* entre une clause c_i contenant le littéral x et c_j une clause contenant l'opposé de ce même littéral $\neg x$. En d'autres termes, $\eta[x, c_i, c_j] = c_i \cup c_j \backslash \{x, \neg x\}$.

2.2 Recherche DPLL

DPLL [11] est une procédure de recherche de type *backtrack*; À chaque nœud les littéraux affectés (le littéral de décision et les littéraux propagés) sont étiquetés avec le même *niveau de décision*, initialisé à 1 et incrémenté à chaque nouveau point de décision. Le niveau de décision courant est le niveau le plus élevé dans la pile de propagation. Lors d'un retour-arrière ("*backtrack*"), les variables ayant un niveau supérieur au niveau du backtrack sont défaites et le niveau de décision courant est décrémenté en conséquence (égal au niveau du backtrack). Au niveau i, l'interprétation partielle courante ρ peut être représentée comme une séquence de décision-propagations de la forme $\langle (x_k^i), x_{k_1}^i, x_{k_2}^i, \ldots, x_{k_{n_k}}^i \rangle$ où le premier littéral x_k^i correspond au littéral de décision x_k affecté au niveau i et chaque $x_{k_j}^i$ de l'ensemble $1 \leq j \leq n_k$ représente les littéraux unitaires propagés à ce même niveau i. Soit $x \in \rho$, On note $l(x)$ le niveau d'affectation de x. Pour une clause α, $l(\alpha)$ est défini comme le maximum des niveaux de ses littéraux affectés.

2.3 Fonctions Booléennes

Une fonction Booléenne est une expression de la forme $y = f(x_1, \ldots, x_n)$, où $f \in \{\vee, \wedge\}$ et où y et x_i sont des littéraux, qui est définie comme suit :

- $y = \wedge(x_1, \ldots, x_n)$ représente l'ensemble de clauses $\{y \vee \neg x_1 \vee \cdots \vee \neg x_n, \neg y \vee x_1, \ldots, \neg y \vee x_n\}$, traduisant la condition selon laquelle la valeur de vérité de y est determinée par la conjonction des valeurs de vérité des x_i t.q.$i \in [1 \ldots n]$;
- $y = \vee(x_1, \ldots, x_n)$ représente l'ensemble de clauses $\{\neg y \vee x_1 \vee \cdots \vee x_n, y \vee \neg x_1, \ldots, y \vee \neg x_n\}$;

Dans la suite, nous considérons seulement les fonctions booléennes de la forme $y = \vee(x_1, \ldots, x_n)$ où y et x_i sont des variables booléennes de la formule originale. En effet, une fonction booléenne $y = \wedge(x_1, \ldots, x_n)$ peut être écrite comme $\neg y = \vee(\neg x_1, \ldots, \neg x_n)$.

Pour une fonction boolénne de la forme $y = f(x_1^{'}, \ldots, x_n^{'})$, où $x_i^{'} \in \{x_i, \neg x_i\}$, la variable y est une variable de sortie tandis que les variables $x_1, \ldots, x_n$ sont appelées variables d'entrée. Ces fonctions nous permettent de détecter un sous-ensemble de variables dépendantes (i.e. variables de sortie) dont la valeur de vérité dépend des valeurs de vérité des variables d'entrée.

2.4 Detection des fonctions booléennes dans une formule CNF

Etant donné une formule CNF $\mathcal{F}$, On peut utiliser deux méthodes différentes pour la détection des fonctions booléennes encodées par les clauses de $\mathcal{F}$.
La première méthode de détection, appelée méthode syntaxique, est une approche de type *pattern matching* qui nous permet de detecter les fonctions booléennes qui apparaissent directement dans la structure de la formule CNF [24].

Exemple 1 *Soit* $\mathcal{F} \supseteq \{(y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3), (\neg y \vee x_1), (\neg y \vee x_2), (\neg y \vee x_3)\}$.

Dans cet exemple, nous pouvons détecter syntaxiquement la fonction $y = \wedge(x_1, x_2, x_3)$.

La seconde méthode est l'approche de détection sémantique où les fonctions sont détectées en utilisant la propagation unitaire (UP) [17]. En effet, cette méthode nous permet de détecter les fonctions booléennes cachées.

Exemple 2 *Let* $\mathcal{F} \supseteq \{(y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3), (\neg y \vee x_1), (\neg x_1 \vee x_4), (\neg x_4 \vee x_2), (\neg x_2 \vee x_5), (\neg x_4 \vee \neg x_5 \vee x_3)\}$.

Dans cet exemple, $UP(\mathcal{F} \wedge y) = \{x_1, x_4, x_2, x_5, x_3\}$ l'ensemble des littéraux obtenus par propagation unitaire (UP) et nous avons la clause $c = (y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3) \in \mathcal{F}$ qui est telle que $c \backslash \{y\} \subset \overline{UP(\mathcal{F} \wedge y)}$. Ainsi, nous pouvons detecter la fonction booléenne $y = \wedge(x_1, x_2, x_3)$, que nous ne pouvons pas détecter en utilisant la méthode syntaxique ci-dessus.

Dans la suite, étant donnée une formule CNF $\mathcal{F}$, nous utilisons ces deux méthodes pour détecter toutes les fonctions booléennes encodés par les clauses de $\mathcal{F}$. De toute évidence, l'ensemble des fonctions booléennes qui peuvent être détectées par l'approche sémantique est une extension de l'ensemble des fonctions booléennes extraites à l'aide de la l'approche syntaxique.

Ces fonctions sont ensuite utilisées pour réduire la taille des différentes clauses apprises durant la recherche. Dans notre implémentation, nous exploitons les deux approches de détections des fonctions booléennes proposées dans [24] et [17].

2.5 Analyse de conflit et graphe d'implications

Le graphe d'implications est une représentation standard utilisée classiquement pour analyser les conflits dans les solveurs **SAT** modernes. À chaque fois qu'un littéral y est propagé, on sauvegarde une référence de la clause impliquant la propagation de y, qu'on note $imp(y)$. La clause $imp(y)$, appelée implication de y, est dans ce cas de la forme $(x_1 \vee \cdots \vee x_n \vee y)$ où chaque littéral x_i est faux sous l'affectation partielle courante $(\rho(x_i) = \mathit{false}, \forall i \in 1..n)$, alors que $\rho(y) = \mathit{vraie}$. lorsqu'un littéral y n'est pas obtenu par propagation mais issue d'une décision, $imp(y)$ est indéfinie, qu'on note par convention $imp(y) = \bot$. Quand $imp(y) \neq \bot$, on note par $exp(y)$ l'ensemble $\{\overline{x} \mid x \in imp(y) \setminus \{y\}\}$, appelé ensemble des *explications* de y. Quand $imp(y)$ est indéfini, on défini $exp(y)$ comme l'ensemble vide.

Définition 1 (Graphe d'implications) *Soit $\mathcal{F}$ une formule CNF, ρ une affectation partielle, et soit exp l'ensemble des explications pour les littéraux déduits (propagés unitairement) dans ρ. Le graphe d'implications associé à $\mathcal{F}$, ρ et exp est $\mathcal{G}^{\rho,exp}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$ où :*

- *$\mathcal{N} = \rho$, i.e., il existe un nœud pour chaque littéral de décision ou propagé ;*
- *$\mathcal{E} = \{(x, y) \mid x \in \rho, y \in \rho, x \in \exp(y)\}$*

Dans le reste du papier, pour des raisons de simplicité, un graphe d'implication est tout simplement noté $\mathcal{G}^{\rho}_{\mathcal{F}}$. On note aussi m le niveau du conflit.

Exemple 3 *$\mathcal{G}^{\rho}_{\mathcal{F}}$, montré dans Figure 1 représente le graphe d'implications de la formule $\mathcal{F}$ et l'affectation partielle ρ donnée ci-dessous : $\mathcal{F} \supseteq \{c_1, \ldots, c_{12}\}$*

$(c_1) \neg x_1 \vee \neg x_{11} \vee x_2$	$(c_2) \neg x_1 \vee x_3$
$(c_3) \neg x_2 \vee \neg x_{12} \vee x_4$	$(c_4) \neg x_1 \vee \vee \neg x_3 \vee x_5$
$(c_5) \neg x_4 \vee \neg x_5 \vee \neg x_6 \vee x_7$	$(c_6) \neg x_5 \vee \neg x_6 \vee x_8$
$(c_7) \neg x_7 \vee x_9$	$(c_8) \neg x_5 \vee \neg x_8 \vee \neg x_9$
$(c_9) \neg x_{10} \vee \neg x_{17} \vee x_1$	$(c_{10}) \neg x_{13} \vee \neg x_{14} \vee x_{10}$
$(c_{11}) \neg x_{13} \vee x_{17}$	$(c_{12}) \neg x_{15} \vee \neg x_{16} \vee x_{13}$

$\rho = \{\langle \ldots x^1_{15} \ldots \rangle \langle (x^2_{11}) \ldots \ldots \rangle \langle (x^3_{12}) \ldots x^3_6 \ldots \rangle \langle (x^4_{14}), \ldots \rangle \langle (x^5_{16}), x^5_{13}, \ldots \rangle\}$. Le niveau du conflit est 5 et $\rho(\mathcal{F}) = false$.

Définition 2 (Clause assertive) *Une clause c de la forme $(\alpha \vee x)$ est appelée clause assertive ssi $\rho(c) = \mathit{false}$, $l(x) = m$ et $\forall y \in \alpha, l(y) < l(x)$. x est appelé littéral assertif.*

L'analyse de conflits est le résultat de l'application de la résolution à partir de la clause conflit en utilisant différentes implications encodées dans le graphe d'implications. On appelle celà dérivation de la clause assertive (DCA en court).

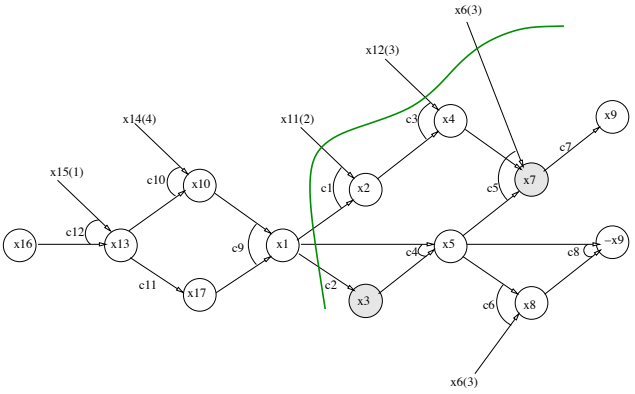


FIGURE 1 – Graphe d'implications $\mathcal{G}^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$

Définition 3 (dérivation de la clause assertive) *La dérivation de la clause assertive pi est une séquence de clauses $\langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ satisfaisant les conditions suivantes :*

1. *$\sigma_1 = \eta[x, imp(x), imp(\neg x)]$, tel que $\{x, \neg x\}$ est le conflit.*
2. *σ_i, pour $i \in 2..k$, est construite en selectionnant un littéral $y \in \sigma_{i-1}$ pour lequel $imp(\overline{y})$ est défini. Nous avons alors $y \in \sigma_{i-1}$ et $\overline{y} \in imp(\overline{y})$: les deux clauses résolues. La clause σ_i est définie comme $\eta[y, \sigma_{i-1}, imp(\overline{y})]$;*
3. *σ_k est en plus une clause assertive.*

considérons à nouveau l'exemple 3. Le parcours du graphe $\mathcal{G}^{\rho}_{\mathcal{F}}$ (voir Fig. 1) conduit à la dérivation de la clause assertive : $\langle \sigma_1, \ldots, \sigma_7 \rangle$ où $\sigma_1 = \eta[x_9, c_7, c_8] = (\neg {x_5}^5 \vee \neg {x_7}^5 \vee \neg {x_8}^5)$ et $\sigma_7 = (\neg {x_{11}}^2 \vee \neg {x_{12}}^3 \vee \neg {x_6}^3 \vee \neg {x_1}^5)$. La clause σ_7 est la prémière clause rencontrée qui contient un seul littéral du niveau de décision courant. Notons que cette résolvante est fausse sous l'affectation ρ. Par conséquent, le littéral $\neg x_1$ est impliqué au niveau 3. Les solveurs SAT ajoutent la clause (σ_7) à la base des clauses apprises, retourne au niveau 3 et affecte le littéral assertif $\neg x_1$ à la valeur de vérité vraie. Le nœud x_1 correspondant au littéral assertif $\neg x_1$ est appelé *First Unique Implication Point (First UIP)*. Pour plus d'informations autours des solveurs CDCL basés sur les clauses apprises, nous reférons les lecteurs à [22, 23, 2].

3 Substitution dynamique des fonctions booléennes

Dans cette section, nous montrons comment les fonctions booléennes détectées dans la formule origi-

nale peuvent être utilisées pour réduire dynamiquement la taille des différentes clauses apprises.

3.1 Exemple

Nous illustrons cette nouvelle approche de substitution dynamique des fonctions booléennes en utilisant un simple exemple.

Exemple 4 *Soit $\mathcal{F}$ la formule CNF précédente, supposons que $\mathcal{F}$ contient aussi les clauses suivantes $\{c_{13}, \ldots, c_{16}\}$: $\mathcal{F} \supseteq \{c_{13}, \ldots, c_{16}\}$*

$(c_{13}) \neg y \vee \neg x_{11} \vee \neg x_{12} \vee \neg x_6$
$(c_{14}) \quad y \vee x_{11}$
$(c_{15}) \quad y \vee x_{12}$
$(c_{16}) \quad y \vee x_6$

Soit $\sigma_7 = (\neg {x_{11}}^2 \vee \neg {x_{12}}^3 \vee \neg {x_6}^3 \vee \neg {x_1}^5)$ la clause apprise précédente après l'analyse de conflit. Les clauses c_{13}, c_{14}, c_{15} et c_{16} encodent la foncion booléenne $y = \vee(\neg x_{11}, \neg x_{12}, \neg x_6)$. Comme on peut le voir, la clause apprise σ_7 contient les entrées de la fonction boolénne précédente. Par conséquent, on peut remplacer ces entrées par la sortie y dans σ_7. Cette substitution permet d'éliminer les littéraux $\neg x_{11}, \neg x_{12}, \neg x_6$ de la clause apprise σ_7.
Après ce processus, on obtient $\sigma'_7 = (\neg x_1 \vee y)$. Il est important de noter que la résolvante σ'_7 peut être obtenue par hyper résolution binaire entre les clauses binaires c_{14}, c_{15} et c_{16} et la clause c_{13}. La principale différence est que dans notre approche, y doit être la sortie d'une fonction booléenne.

Exemple 5 *Soit $\mathcal{F}$ la formule CNF précédente, supposons maintenant que $\mathcal{F}$ contient plutôt les clauses suivantes $\{c_{13}, \ldots, c_{17}\}$: $\mathcal{F} \supseteq \{c_{13}, \ldots, c_{17}\}$*

$(c_{13}) \quad y \vee \neg x_{12} \vee \neg x_6$
$(c_{14}) \quad \neg y \vee x_{14}$
$(c_{15}) \quad \neg y \vee x_{15}$
$(c_{16}) \quad \neg x_{14} \vee \neg x_{15} \vee x_{12}$
$(c_{17}) \quad \neg x_{12} \vee \neg x_{14} \vee x_6$

Dans ce deuxième exemple, on détecte par propagation unitaire la fonction booléenne $y = \wedge(x_{12}, x_6)$ qui est équivalente à $\neg y = \vee(\neg x_{12}, \neg x_6)$. En effet, $UP(\mathcal{F} \wedge y) = \{x_{14}, x_{15}, x_{12}, x_6\}$. Les littéraux $\neg x_6$ et $\neg x_{12}$ sont éliminés de la clause apprise σ_7 et remplaçés par la sortie $\neg y$. Après cette substitution, on obtient $\sigma''_7 = (\neg y \vee \neg x_{11} \vee \neg x_1)$.

La méthode de détection basée sur la propagation unitaire nous permet de détecter toutes les fonctions booléennes que nous ne pouvons pas détecter syntaxiquement. Il est important de noter que lorsque les fonctions booléennes sont détectées en utilisant la propagation unitaire, la résolvante σ''_7 ne peut pas être obtenue directement par hyper-résolution binaire. Pour que cela soit possible, on a besoin premièrement de générer les clauses $(\neg y \vee x_{12})$ et $(\neg y \vee x_6)$ et par résolution avec les clauses c_{14}, c_{15}, c_{16} et c_{17}.

Notons que plusieurs entrées peuvent être substituées par la même sortie. Ceci arrive lorsque nous avons plusieurs fonctions booléennes dans la formule originale avec la même sortie. Dans ce cas, toutes les différentes entrées incluses dans la clause apprise sont éliminées mais cette sortie n'est ajoutée qu'une seule fois dans la clause apprise.

Les fonctions booléennes utilisées ici pour réduire les clauses apprises sont encodées par les clauses de la formule originale. Ainsi, aucune variable additionnelle n'est ajoutée à la formule comme dans le cas des systèmes de preuves par résolution étendue. Cependant, certaines fonctions booléennes encodées dans la formule CNF sont introduites par le principe d'extension utilisé dans la phase d'encodage CNF.

3.2 Substitution dynamique

On donne à présent une présentation formelle de notre approche basée sur la substitution dynamique.

Proposition 1 *[Substitution dynamique] Soit $\mathcal{F}$ une formule CNF, $\sigma = (\alpha \vee x)$ une clause apprise, où x est le littéral assertif et α une disjonction de littéraux. Soit $y = \vee(x_1, \ldots, x_n)$ avec $n \geq 2$, une fonction booléenne encodée par les clauses de $\mathcal{F}$ t.q. $\{x_1, \ldots, x_n\} \subseteq \alpha$, alors σ peut être réduite à σ' en substituant $\{x_1, \ldots, x_n\}$ par y dans α t.q. :*
- *$\sigma' = \beta \vee x$ si $y \in \alpha$*
- *$\sigma' = \beta \vee x \vee y$ sinon*
  avec $\beta = \alpha \backslash \{x_1, \ldots, x_n\}$

Propriété 1 *Soit $\mathcal{F}$ une formule CNF, $\sigma = (\alpha \vee x)$ une clause apprise, où x est le littéral assertif. Soit $y = \vee(x_1, \ldots, x_n)$ une fonction booléenne encodée par les clauses de $\mathcal{F}$ t.q.$\{x_1, \ldots, x_n\} \subseteq \alpha$. on a :*
- *$\forall 1 \leq i \leq n, \rho(x_i) = faux$ et $\rho(y) = faux$*
- *$l(y) = max\{l(x_i), 1 \leq i \leq n\}$ et $l(y) < l(x)$*

Preuve 1 *Par définition de la clause apprise σ, on a $\rho(\sigma) = faux$. Comme $\{x_1, \ldots, x_n\} \subseteq \sigma$. Par conséquent, $\forall 1 \leq i \leq n, \rho(x_i) = faux$.*
On a $y = \vee(x_1, \ldots, x_n)$ alors $\rho(y) = faux$.
Par définition de la fonction booléenne $y = \vee(x_1, \ldots, x_n)$, $l(y)$ est le niveau du dernier littéral x_i qui fut affecté à faux. Par conséquent, $l(y) =$

$max\{l(x_i), 1 \leq i \leq n\}$. *Comme* x *est le littéral assertif, par définition,* $\forall z \in \alpha, l(z) < l(x)$. *Puisque* $\{x_1, \ldots, x_n\} \subseteq \alpha$ *et* $l(y) = max\{l(x_i), 1 \leq i \leq n\}$, *alors* $l(y) < l(x)$.

Propriété 2 *Soit* $\sigma = (\alpha \vee x)$ *une clause apprise, et* $G = \{g_1, \ldots, g_n\}$ *un ensemble de fonctions booléennes détectées dans la formule originale telle que les différents ensembles de variables d'entrée associées aux différentes fonctions booléennes dans* G *soient tous disjoints et tous inclus dans* α. *Soit* $\sigma_1, \ldots, \sigma_n$ *la séquence de clauses apprises obtenues successivement après chaque substitution t.q.à l'étape* i, σ_i *est obtenue en remplaçant un ensemble de variables d'entrée dans* σ_{i-1} *par une variable de sortie* y_j, *alors* $|\sigma_i| < |\sigma_{i-1}|$, *et* $|\sigma_i| > 1$, $1 \leq i, j \leq n$

Preuve 2 *Soit* σ_0 *la clause initialement apprise. Comme* σ_i *est obtenue par substitution d'au moins deux littéraux (entrée d'une fonction booléenne) dans* σ_{i-1} *par un littéral qui est la sortie de cette fonction, on a* $|\sigma_i| < |\sigma_{i-1}|$, *pour* $1 \leq i \leq n$. *De plus,* $\forall 1 < i < n$, $x \in \sigma_i$, *où* x *est le littéral assertif et n'est pas considéré dans la substitution, alors* $|\sigma_i| > 1$, $1 \leq i \leq n$

Propriété 3 *Etant donné une clause apprise* σ, *et un ensemble* $\mathcal{G} = \{g_1, \ldots, g_n\}$ *de fonctions booléennes détectées dans la formule CNF originale. La complexité temporelle de notre approche de substitution dynamique est quadratique dans le pire des cas.*

Preuve 3 *On a* $|\mathcal{G}|$ *fonctions booléennes. Pour chaque fonction booléenne, on fait* $|\sigma|$ *comparaisons. Ainsi, on peut faire au plus* $|\mathcal{G}| \times |\sigma|$ *comparaisons pour réaliser la substitution.*

4 Expérimentations

Les expérimentations menées sont effectuées sur un large panel d'instances industrielles et crafted provenant de la compétition SAT 2009 et de la SAT-Race 2009. Toutes les instances sont simplifiées par prétraitement `SatElite` [12]. On peut noter que `SatElite` substitue certaines fonctions booléennes à l'étape de pré-traitement. Par conséquent, les fonctions booléennes détectées sont celles qui restent dans la formule simplifiée par `SatElite`. Nous avons intégré notre approche de substitution dynamique dans `Minisat` 2.2 [13] et nous avons effectué une comparaison entre les résultats obtenus par les solveurs originaux et ceux incluant l'approche de substitution dynamique. Il est aussi important de noter le détail d'implémentation suivant : Quand les variables d'entrée sont substituées par la variable de sortie, l'activité de la variable de sortie est mise à jour de la même manière que les autres variables de la clause apprise. Ceci permet de focaliser la recherche sur certaines variables de sortie. Ce détail d'implémentation est similaire à celui implémenté dans [3].

Tous les tests sont effectués sur un cluster Xeon 3.2GHz (2 GB RAM). Les résultats du temps de calcul sont indiqués en secondes.
Pour ces expérimentations, le temps de calcul limite est fixé à 1 heure. les différentes tables montrent un ensemble représentatif de familles d'instances.

4.1 Problèmes industriels

Les tables 1 et 2 détaillent les résultats sur les problèmes SAT industriels issus de la compétition SAT'2009 et SAT-Race'2010 en utilisant respectivement les méthodes de détection syntaxique et sémantique pour la détection des fonctions booléennes dans la formule originale.

Pour chaque instance, nous reportons son nom (Instance), son nombre de variables et de clauses (#Var, #Cl), le temps utilisé par `Minisat` pour résoudre l'instance, le temps utilisé par `Minisat+DS` pour résoudre l'instance, le nombre total de substitution éffectuée ($nbSub$) et le nombre total de littéraux éliminés par la substitution des entrées par les sorties ($totalSub$).

La table 1 représente un focus sur quelques familles industrielles. Sur ces familles, les améliorations sont relativement importantes. Par exemple, si on considère la famille vmpc, on peut voir que notre substitution dynamique permet d'avoir un gain d'un ordre de magnitude (instances 24 and 27). Sur la même famille, `Minisat+DS` résoud 2 instances de plus que `Minisat` (instances 31 and 33).
Sur la famille gss, sur les 7 instances résolus par `Minisat+DS` et `Minisat`, `Minisat+DS` est meilleur sur 5 instances. Sur la même famille, `Minisat+DS` résoud une instance de plus que `Minisat` (instance 23).

En résumé, on peut voir que sur certaines familles, `Minisat+DS` est plus rapide et résoud plus de problèmes que `Minisat`.

4.2 Problèmes crafted

Ces problèmes sont conçus à la main et beaucoup d'entre eux sont conçus dans le but de battre tous les solveurs DPLL existants. Ils contiennent par exemple les instances Quasi-group, instances SAT forcés aléatoires, counting, instances "ordering" et "pebbling", problèmes sociaux du golfeur, etc.

La table 3 montre que sur les 11 instances résolues de la famillle QG, `Minisat+DS` résoud plus efficacement

7 instances. Sur la famille rbsat, `Minisat` est meilleur sur 5 instances parmi les 8 instances résolues. On peut remarquer que sur 4 de ces 5 instances, la valeur de nbSum et totalSub est égal à 0 ce qui peut expliquer les limites de l'efficacité de notre approche dans ces cas.

Globalement, on peut voir que l'ajout de notre processus de substitution dynamique à `Minisat` améliore ses performances sur certaines familles de problèmes crafted.

La table 4 montre que ces performances sont encore améliorées en utilisant la méthode sémantique pour la détection des fonctions booléennes. Dans cette table, sur la famille QG, les temps de résolution de `Minisat+DS` ont été améliorés de manière significative.

4.3 Summary

Dans l'ensemble, nos expérimentations nous ont permis de démontrer deux choses. Premièrement, notre technique ne dégrade pas mais au contraire améliore souvent les performances des solveurs DPLL sur les problèmes industriels et crafted. Second, elle améliore l'applicabilité de ces algorithmes sur des classes de problèmes qui sont faites pour être difficiles pour eux. Les résultats présentés dans les tables précédentes montre que notre approche est éfficace sur certaines familles d'instances SAT. Globalement, sur l'ensemble d'instances, notre approche est compétitive et résoud un nombre similaire d'instances.

A partir de ces expérimentations, on observe que notre méthode présente certaines complémentarités avec les solveurs SAT classique tels que `Minisat`. En effet, on observe que plusieurs instances sont résolues uniquement par notre approche tandis que d'autres ne sont résolues que par `Minisat`. Sur de nombreux cas où notre approche dégrade, le nombre de substitution est assez faible.

5 Remerciements

Ce travail a été soutenu par l'Agence Nationale de la Recherche - projet ANR programme blanc TUPLES.

6 Conclusion

Ce papier présente une nouvelle technique de substitution dynamique de fonctions booléennes détectées dans une formule booléenne sous forme CNF. Cette approche peut être vue comme une façon originale de manipuler les sous-formules représentées par ces fonctions. Elle nous permet d'imiter le principe de résolution étendue. En effet, notre approche exploite les fonctions booléennes cachées généralement introduites pendant la phase d'encodage CNF, et les utilise pour minimiser les clauses apprises. Ce qui nous permet d'exploiter ces variables auxiliaires au lieu d'en ajouter des variables supplémentaires au cours de la phase de résolution. La substitution des variables d'entrée par les variables de sortie au cours de l'analyse des conflits, est une façon élégante de manipuler ces variables de sortie. Les résultats expérimentaux sur certaines classes d'instances SAT, montrent clairement que notre approche est compétitive avec l'état de l'art des solveurs SAT, bien qu'il présente certains aspects de complémentarité. De plus, notre approche présente des améliorations intéressantes sur certaines familles d'instances SAT.

Comme travaux futures, nous envisageons d'étendre notre approche en visant une substitution partielle. En effet, même si l'ensemble des variables d'entrée apparaît partiellement dans la clause apprise donnée, on peut toujours substituer ces variables par la variable de sortie correspondante. Dans un tel cas, il pourrait être intéressant de calculer la substitution qui maximise la réduction de la taille de la clause apprise.

Références

[1] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *Transactions on Computer Aided Design*, 2003.

[2] G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Saïs. Generalized framework for conflict analysis. In *Proceedings of the eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, pages 21–27, 2008.

[3] Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning sat solvers. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI'10)*, 2010.

[4] Fahiem Bacchus. Enhancing davis putnam with extended binary clause reasoning. In *AAAI/IAAI*, pages 613–619, 2002.

[5] Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In *SAT*, pages 341–355, 2003.

[6] Roberto J. Bayardo, Jr. and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, 1997.

Instance	(#Var, #Cl)	Statut	Minisat	Minisat + DS		
			time(s)	time(s)	nbSub	totalSub
vmpc_24	(576 , 67872)	SAT	18.41	**4.49**	9	198
vmpc_25	(625 , 76775)	SAT	**19.89**	57.37	134	3082
vmpc_26	(676 , 86424)	SAT	**14.31**	76.46	267	6408
vmpc_27	(729 , 96849)	SAT	43.66	**10.06**	24	600
vmpc_28	(784 , 108080)	SAT	96.72	**62.55**	189	4914
vmpc_29	(841 , 120147)	SAT	**70.34**	755.24	1220	32994
vmpc_30	(900 , 133080)	SAT	**476.82**	877.05	1223	34244
vmpc_31	(961 , 146909)	SAT	timeout	**3366.28**	2915	84564
vmpc_33	(1089 , 177375)	SAT	timeout	**3151.05**	2340	72540
vmpc_34	(1156 194072)	SAT	**timeout**	**timeout**	2048	65568
gss-16-s100	(31248 , 93904)	SAT	18.04	**6.15**	986	1132
gss-20-s100	(31503 , 94748)	SAT	1275.94	**7.22**	646	736
gss-23-s100	(31711 , 95400)	SAT	timeout	**2213.94**	118104	136100
gss-17-s100	(31318 , 94116)	SAT	107.71	**40.26**	3706	4949
gss-19-s100	(31435 , 94548)	SAT	152.60	**105.78**	7533	8797
gss-13-s100	(30867 , 92735)	SAT	6.21	**5.78**	366	466
gss-14-s100	(31229 , 93855)	SAT	**4.05**	11.63	612	655
gss-15-s100	(31238 , 93878)	SAT	**12.76**	30.73	3257	4336
sha0_35_1	(48689 , 204053)	SAT	71.23	**23.12**	7421	10241
sha0_36_5	(50073 , 210223)	SAT	471.32	**17.51**	4318	5498
sha0_35_3	(48689 , 204067)	SAT	29.02	**9.78**	1617	1879
sha0_35_2	(48689 , 204071)	SAT	**22.88**	202.25	83757	110310
UTI-20-5p0	(225296 1192799)	UNSAT	timeout	**1705.26**	326191	2062182
rbcl_xits_07	(1128 , 57446)	UNSAT	219.12	**156.02**	112280	372012
uts-l06-ipc5-h35	(175670 837466)	SAT	3.20	**0.84**	5	10
md5_48_1	(66892 , 279248)	SAT	221.94	**145.82**	26021	34469
md5_47_4	(65604 , 273506)	SAT	64.13	**21.60**	3839	4614
md5_48_3.cnf	(66892 , 279258)	SAT	**160.06**	210.04	42255	57303
partial-10-15-s	(261020 , 1211106)	SAT	3039.08	**649.65**	13696	18345
partial-10-13-s	(234673 , 1071339)	SAT	timeout	**1470.39**	31336	43086
rpoc_xits_08	(1278 , 74789)	UNSAT	timeout	**2974.28**	459189	1528680
uts-l06-ipc5-h32	(163755 , 800163)	UNSAT	**10.62**	48.07	392	28040
maxxororand032	(23696 , 70703)	UNSAT	859.81	**542.29**	241743	1051278
manol-pipe-g10bidw	(237485 , 705220)	UNSAT	494.72	**172.32**	40268	75273
manol-pipe-c7nidw	(169072 , 501421)	UNSAT	**33.35**	54.45	39464	60577
simon-s02b-dp11u10	(9197 , 25271)	UNSAT	338.53	**295.11**	350707	421961
mizh-sha0-36-2	(50073 , 210239)	SAT	892.01	**27.77**	8888	11784
mizh-sha0-36-3	(50073 , 210235)	SAT	**110.29**	502.11	239539	341651
mizh-md5-48-5	(66892 , 279256)	SAT	74.91	**10.83**	2523	2902
velev-pipe-sat-1.0-b10	(118040 , 8804672)	SAT	**58.25**	67.40	0	0

TABLE 1 – Instances Industrielles : substitution dynamique et détection par la méthode syntaxique

Instance	(#Var,#Cl)	Statut	Minisat	Minisat + DS		
			time(s)	time(s)	nbSub	totalSub
gss-17-s100	(31318 , 94116)	SAT	107.71	**96.04**	6440	7456
gss-20-s100	(31503 , 94748)	SAT	1275.94	**425.667**	28295	35016
sha0_35_1	(48689 , 204053)	SAT	71.23	**27.67**	9182	12319
sha0_36_5	(50073 , 210223)	SAT	471.32	**130.26**	50706	69396
md5_48_1	(66892 , 279248)	SAT	221.94	**65.12**	11265	14152
c8idw_s	(122321 , 361795)	UNSAT	6.18	**0.41**	239	321
f7nidw	(310434 , 923497)	UNSAT	1867.02	**1217.44**	41689	53540
g7nidw	(75496 , 222379)	UNSAT	62.36	**17.54**	3059	4862
eq.atree.braun.11	(1694 , 5726)	UNSAT	3593.43	**3472.79**	20198398	32717812
rbcl_xits_07	(1128 , 57446)	UNSAT	219.12	**169.61**	104596	346524
rpoc_xits_07	(1128 , 63345)	UNSAT	197.91	**185.48**	86856	282885
gus-md5-09	(69487 , 226581)	UNSAT	204.46	**193.52**	12015	20529
mizh-sha0-36-2	(50073 , 210239)	SAT	892.01	**272.56**	97731	135321
simon-s02b-dp11u10	(9197 , 25271)	UNSAT	338.53	**258.07**	280825	331549
manol-pipe-g7nidw	(75496 , 222379)	UNSAT	62.64	**15.94**	3059	4862
mizh-sha0-36-3	(50073 , 210235)	SAT	110.29	**25.71**	7870	10689
mizh-md5-47-3	(65604 , 273522)	SAT	117.49	**39.97**	8184	10314
schup-l2s-motst-2	(507145 , 1601920)	SAT	11.75	**9.26**	85	246
post-cbmc-aes-ee	(498723 , 2928183)	UNSAT	568.25	**489.90**	429	2173
ndhf_xits_21_SAT	(4466 , 542457)	SAT	1.41	**0.55**	618	16716
post-cbmc-aes-ele	(270963 , 1601376)	UNSAT	7.45	**4.84**	22	66
dated-5-15-u	(151952 , 697321)	UNSAT	**323.40**	418.72	6085	11012
dated-10-13-s	(181082 , 824258)	SAT	**5.69**	20.79	92	149
q_query_3_l48_lambda	(34656 , 174528)	UNSAT	**78.88**	141.38	164871	195089
q_query_3_l45_lambda	(32313 , 161529)	UNSAT	**91.85**	162.14	182561	216837
zfcp-2.8-u2-nh	(10950109, 32697150)	SAT	**27.73**	31.30	0	0
vmpc_25	(625 , 76775)	SAT	**19.89**	49.05	73	1613
mizh-sha0-35-3	(48689 , 204067)	SAT	**28.49**	36.87	11529	18419
mizh-sha0-36-4	(50073 , 210235)	SAT	**89.19**	111.12	44991	64725
goldb-heqc-term1mul	(3504 , 22229)	UNSAT	**42.02**	69.53	17	51
countbitssrl032	(18607 , 55724)	UNSAT	**646.13**	903.15	135175	171439

TABLE 2 – Instances Industrielles : substitution dynamique et détection par la méthode sémantique

Instance	(#Var,#Cl)	Statut	Minisat	Minisat + DS		
			time(s)	time(s)	nbSub	totalSub
QG7a-gensys-icl004	(2401 , 15960)	UNSAT	523.70	**401.41**	672102	3433306
QG6-gensys-icl001	(1587 , 8013)	UNSAT	245.77	**218.43**	856139	4361401
QG6-gensys-icl004	(1605 , 8025)	UNSAT	189.05	**93.22**	454510	2154537
QG8-gensys-ukn005	(1133 , 56210)	UNSAT	51.20	**35.32**	10267	70734
QG7a-gensys-ukn005	(2765 , 18046)	SAT	47.53	**26.45**	55066	313842
QG-gensys-icl003	(1472 , 7737)	UNSAT	134.79	**87.20**	409490	1884892
QG7a-gensys-ukn001	(2737 , 18375)	SAT	42.38	**9.49**	23785	137159
QG7-gensys-icl006	(728 , 17199)	UNSAT	**1311.19**	1483.74	292722	1388841
QG7-dead-dnd005	(502 , 11816)	UNSAT	**431.93**	696.91	220929	1119611
QG6-dead-dnd002	(1339 , 7095)	UNSAT	**319.51**	418.85	1574081	8255326
QG-gensys-brn008	(1467 , 752)	UNSAT	**98.60**	105.40	487138	2224629
rbsat-v760c43649g8	(760 , 43649)	SAT	39.20	**29.59**	6	114
rbsat-v760c43649gyes4	(760 , 43649)	SAT	7.29	**3.30**	1	23
rbsat-v760c43649gyes6	(760 ,43649)	SAT	754.22	**118.22**	241	4338
rbsat-v760c43649gyes1	(760 , 43649)	SAT	**2481.48**	2499.2	0	0
rbsat-v760c43649g4	(760 , 43649)	UNSAT	**546.31**	622.815	0	0
rbsat-v760c43649g1	(760 , 43649)	SAT	**178.13**	197.923	0	0
rbsat-v760c43649g10	(760 , 43649)	SAT	**68.83**	121.836	78	1658
rbsat-v945c61409gyes9	(945 , 61409)	SAT	**824.15**	831.814	0	0
rbsat-v945c61409gyes4	(945 , 61409)	SAT	timeout	**3453.15**	241	5613
mod3block_2vars_11	(526 , 146535)	SAT	timeout	**1488.51**	340924	355550
mod4block_2vars_10	(479 , 123509)	SAT	2393.86	**679.48**	12822	26072
em_7_4_8_all	(1617 , 37237)	SAT	134.58	**8.49**	1016	6362
em_8_4_5_all	(2420 , 62900)	SAT	**32.08**	52.32	16490	61304
em_11_3_4_all	(8713 , 414651)	SAT	**14.31**	87.01	5785	15145
em_7_4_8_exp	(1617 , 25837)	SAT	194.67	**43.15**	2822	19580
em_7_3_6_exp	(1473 , 22887)	SAT	**2.46**	17.86	4007	17174
em_7_3_6_fbc	(1473 , 19063)	SAT	1702.68	**101.68**	42879	186028
em_8_4_5_fbc	(2420 , 33572)	SAT	timeout	**1012.14**	445179	1547530
em_7_3_6_cmp	(1473 , 11563)	SAT	761.50	**299.31**	72641	245066
em_8_4_5_cmp	(2420 , 22340)	SAT	1127.77	**782.49**	355928	1144871
em_7_4_8_cmp	(1617 , 14513)	SAT	**2783.97**	timeout	621973	2334680
instance_n9_i9_pp_ci_ce	(33867 , 457280)	SAT	timeout	**1482.43**	519105	907405
instance_n9_i9_pp	(33867 , 454832)	SAT	640.35	**339.55**	144223	234487
instance_n5_i6_pp_ci_ce	(4380 , 31980)	UNSAT	**1.27**	1.58	700	1621
instance_n8_i8_pp_ci_ce	(21640 , 257551)	SAT	523.53	**253.46**	128221	213622
strips-gripper-18t35	(11318 , 316793)	SAT	36.08	**16.15**	16	16

TABLE 3 – Instances crafted : substitution dynamique et détection par la méthode syntaxique

Instance	(#Var,#Cl)	Statut	Minisat	Minisat + DS		
			time(s)	time(s)	nbSub	totalSub
QG7a-gensys-icl004	(2401 , 15960)	UNSAT	523.70	**352.33**	672102	3433306
QG6-gensys-ukn003	(2123 , 9177)	UNSAT	492.87	**448.36**	1249830	6229729
rbsat-v945c61409gyes9	(945 , 61409)	SAT	824.15	**774.67**	0	0
QG6-gensys-brn007	(1477 , 7809)	UNSAT	119.94	**103.15**	444274	2185145
QG7-gensys-icl006	(728 , 17199)	UNSAT	**1311.19**	1447.36	292722	1388841
QG7-dead-dnd005	(502 , 11816)	UNSAT	**431.93**	682.53	220929	1119611
QG6-gensys-icl001	(1587 , 8013)	UNSAT	245.77	**215.35**	856139	4361401
QG6-gensys-icl004	(1605 , 8025)	UNSAT	189.05	**91.66**	454510	2154537
QG8-gensys-ukn005	(1133 , 56210)	UNSAT	51.20	**30.95**	10267	70734
QG7a-gensys-ukn005	(2765 , 18046)	SAT	47.53	**26.39**	55066	313842
QG-gensys-icl003	(1472 , 7737)	UNSAT	134.79	**85.74**	409490	1884892
QG7a-gensys-ukn001	(2737 , 18375)	SAT	42.38	**9.35**	23785	137159
mod4block_2vars_11	(530 , 153478)	SAT	629.84	**120.62**	7769	15662
mod3block_2vars_11	(526 , 146535)	SAT	timeout	**1362.28**	340924	355550
mod3_4vars_6gates	(289 , 33900)	UNSAT	**517.72**	540.37	7704	11264
mod4block_2vars_10	(479 , 123509)	SAT	2393.86	**638.17**	12822	26072
rbsat-v760c43649g8	(760 , 43649)	SAT	39.20	**28.33**	6	114
rbsat-v760c43649g10	(760 , 43649)	SAT	**68.83**	116.14	78	1658
rbsat-v760c43649gyes4	(760 , 43649)	SAT	7.29	**3.26**	1	23
rbsat-v760c43649gyes6	(760 , 43649)	SAT	754.22	**111.77**	241	4338
rbsat-v760c43649gyes1	(760 , 43649)	SAT	**2481.48**	346.825	52	1144
rbsat-v945c61409gyes4	(945 , 61409)	SAT	timeout	**3217.75**	241	5613
rbsat-v760c43649gyes1	(760 , 43649)	SAT	2481.48	2363.94	0	0
em_7_4_8_all	(1617 , 37237)	SAT	134.58	**8.49**	1016	6362
em_7_4_8_exp	(1617 , 25837)	SAT	194.67	**41.18**	2822	19580
em_8_4_5_all	(2420 , 62900)	SAT	**32.08**	50.10	16490	61304
em_11_3_4_all	(8713 , 414651)	SAT	**14.31**	86.03	5785	15145
em_12_2_4_all	(12584 , 729136)	SAT	**24.86**	664.82	30255	79505
em_7_3_6_fbc	(1473 , 19063)	SAT	1702.68	**97.85**	42879	186028
em_8_4_5_fbc	(2420 , 33572)	SAT	timeout	**953.59**	445179	1547530
em_7_3_6_cmp	(1473 , 11563)	SAT	761.50	**290.91**	72641	245066
em_9_3_5_exp	(3857 , 108164)	SAT	**42.82**	376.20	140442	517796
em_8_4_5_cmp	(2420 , 22340)	SAT	1127.77	**725.73**	355928	1144871
instance_n9_i9_pp_ci_ce	(33867 , 457280)	SAT	timeout	**1476.45**	519105	907405
instance_n9_i9_pp	(33867 , 454832)	SAT	640.35	**335.01**	144223	234487
instance_n8_i8_pp_ci_ce	(21640 , 257551)	SAT	523.53	**235.77**	128221	213622
sgen1-unsat-85-100	(85 , 180)	UNSAT	**499.61**	596.44	0	0
sgen1-unsat-73-100	(73 , 156)	UNSAT	**38.53**	51.36	0	0
sgen1-unsat-97-100	(97 , 204)	UNSAT	**3561.91**	timeout	0	0
sgen1-sat-160-100	(160, 384)	SAT	**44.29**	55.17	0	0
strips-gripper-18t35	(11318 , 316793)	SAT	36.08	**14.53**	16	16
999999000001nw	(4667 , 18492)	UNSAT	1287.23	**1082.17**	12	12

TABLE 4 – Instances Crafted : substitution dynamique et détection par la méthode sémantique

[7] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Understanding the power of clause learning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1194–1201, 2003.

[8] Belaid Benhamou and Lakhdar Saïs. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning*, 12(1) :89–102, February 1994.

[9] Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8 :28–32, October 1976.

[10] James Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Principles of Knowledge Representation and Reasoning (KR'96)*, pages 148–159. 1996.

[11] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397, 1962.

[12] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 61–75, 2005.

[13] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2002.

[14] Pierre Flener, Justin Pearson, Meinolf Sellmann, Pascal Van Hentenryck, and Magnus Ågren. Dynamic structural symmetry breaking for constraint satisfaction problems. *Constraints*, 14(4) :506–538, 2009.

[15] Ian P. Gent and Barbara Smith. Symmetry breaking in constraint programming. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence(ECAI'00)*, pages 599–603, 2000.

[16] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 431–437, 1998.

[17] É. Grégoire, R. Ostrowski, B. Mazure, and L. Saïs. Automatic extraction of functional dependencies. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 122–132, 2004.

[18] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39 :297–308, 1985.

[19] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic restart policies. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, pages 674–682, 2002.

[20] Balakrishnan Krishnamurthy. Shorts proofs for tricky formulas. *Acta Informatica*, 22 :253–275, 1985.

[21] Balakrishnan Krishnamurthy and Robert N. Moll. Examples of hard tautologies in the propositional calculus. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC'81)*, pages 28–37, 1981.

[22] Joao P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, 1996.

[23] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.

[24] R. Ostrowski, É. Grégoire, B. Mazure, and L. Saïs. Recovering and exploiting structural knowledge from cnf formulas. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming(CP'02)*, pages 185–199, Ithaca (N.Y.), 2002.

[25] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning sat solvers with restarts. In *CP*, pages 654–668, 2009.

[26] Jean-Francois Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS'93)*, pages 350–361, 1993.

[27] G.S. Tseitin. On the complexity of derivations in the propositional calculus. In H.A.O. Slesenko, editor, *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

Intensification de la Recherche dans les Solveurs SAT Modernes

Said Jabbour[1] **Jerry Lonlac**[1,2] **Lakhdar Saïs**[1]

[1] CRIL - CNRS - Université Lille-Nord de France F-62307 Lens Cedex
[2] Département d'Informatique - Université de Yaoundé 1 B.P. 812 Yaoundé, Cameroun
{jabbour,lonlac,sais}@cril.fr

Abstract

Les Redémarrage et la recherche basée sur les activités sont deux composantes importantes et liées des Solveurs SAT Modernes. D'une part, la mise à jour des activités des variables impliquées dans l'analyse des conflits vise à circonscrire la partie la plus importante de la formule booléenne. Alors que le redémarrage permet au solveur de réorganiser les variables en focalisant la recherche sur la partie la plus importante de l'espace de recherche. Cette combinaison permet au solveur de diriger la recherche sur la sous-formule la plus contraignante. Dans ce papier, nous proposons de mettre en avant cette recherche basée sur l'intensification en collectant les variables rencontrées au cours de la denière analyse de conflit . A chaque redémarrage, ces variables sont à nouveau sélectonnées en utilisant l'ordre prédefini. Ce simple principe d'intensification apporte des améliorations significatives lorsqu'il est intégré aux solveurs SAT modernes.

1 Introduction

Le problème SAT, i.e., le problème de décider si une formule booléenne sous forme normale conjonctive (CNF) est satisfiable ou non est central en Informatique et en Intelligence Artificielle incluant les problème de satisfaction de contraintes (CSP), planification, raisonnement non-monotone, vérification de logiciels, etc. Aujourd'hui, SAT a gagné une audience considérable avec l'apparition d'une nouvelle génération de solveurs SAT capable de résoudre de très grandes instances issues du codage des applications du monde réel ainsi que par le fait que ces solveurs constituent d'importants composants de base pour plusieurs domaines, e.g., SMT (SAT modulo théorie), démonstration automatique, comptage de modèles, problème QBF, etc. Ces solveurs souvent appelés *Solveurs SAT Modernes* [10, 3], sont basés sur la procédure DPLL classique [2] renforcée avec : (i) une mise en œuvre efficace de la propagation unitaire à travers les structures de données supplémentaires et paresseuses (ii) stratégies de redémarrage [4, 8], (iii) activité basée sur l'heuristique de choix de variables (comme VSIDS) [10], et (iv) *apprentissage de clauses* [9, 10].

Les redémarrage et la recherche basée sur les activités sont deux composantes importantes et liées des Solveurs SAT Modernes. D'une part, la mise à jour des activités des variables impliquées dans l'analyse des conflits vise à circonscrire la partie la plus importante de la formule booléenne. Alors que le redémarrage permet au solveur de réorganiser les variables en focalisant la recherche sur cette sous-formule contraignante. Cette combinaison permet au solveur d'intensifier la recherche et en même temps d'éviter le trashing. Cette forte connexion bien connue entre les redémarrages et l'ordonnancement des variables a aussi une conséquence directe sur la clause apprise. Les effets de redémarrage sur la clause apprise ont été largement étudiés (e.g. [1, 7, 12]). Notre intuition est que si les solveurs SAT sont capables de résoudre efficacement des instances applicatives avec des millions de variables et de clauses, cela signifie que la partie la plus contraignante de la formule (ou sous-ensemble de variables) est de taille raisonnable. Ceci est lié à l'observation faite précédemment sur la taille des ensembles backdoor [13, 14] observé dans plusieurs domaines d'applications.

Dans nos travaux précédents, et dans le solveur parallèle porfolio ManySAT [6], nous avons montré comment les deux principes bien connus de diversification et d'intensification peuvent être combinés dans le contexte de l'architecture Maîtres/Esclaves [5]. Les Maîtres exécutent une stratégie de recherche originale,

en veillant à la diversification, tandis que les unités restantes, classées comme des esclaves sont là pour intensifier la stratégie de leur maître. Par intensification nous voulons dire que l'esclave explorerait différemment autour de l'espace de recherche exploré par le maître.

Dans ce papier, nous proposons de mettre en avant cette recherche basée sur l'intensification en collectant les variables rencontrées au cours de la dernière analyse de conflit. Ainsi, à chaque redémarrage, le solveur se branche en priorité sur ces variables. Ce simple principe d'intensification apporte des améliorations significatives lorsqu'il est integré au solveur SAT Minisat2.2.

2 Intensification de la Recherche

Comme nous avons mentionné dans la section précédente, le lien entre les redémarrages et les heuristiques d'ordonnancement des variables joue un rôle important dans la résolution SAT. Ainsi, l'idée de base de ce papier est focalisée sur les relations entre ces deux composantes. En effet, au cours de la recherche, à chaque conflit, un parcours du graphe d'implications est effectué à partir du conflit jusqu'au dernier UIP (nœud x_{11} - coupe 3 dans Figure 1) et l'ensemble des variables correspondant aux différents nœuds visités sont insérées dans une queue. A chaque redémarrage, le solveur se branche en priorité sur les variables collectées. Lorsque toutes les variables de la queue sont affectées, le solveur suit l'heuristique de branchement ordinaire VSIDS [10]. De cette façon, les variables les plus proches du côté des conflits sont d'abord affectées en utilisant les polarités des littéraux progressivement sauvegardées [11].

Nous illustrons cette nouvelle approche de recherche basée sur l'intensification en utilisant un simple exemple.
Soit $\mathcal{G}^{\rho}_{\mathcal{F}}$ (Figure 1) un graphe d'implications associé à la formule CNF $\mathcal{F}$ et l'affectation partielle ρ. Chaque nœud dans le graphe d'implications correspond à une affectation d'un littéral de décision (nœud x_4, x_{17} et x_{11} affectés respectivement au niveau 2, 3 et 5) ou à un littéral assigné par propagation unitaire. Par exemple le littéral x_{16} est propagé grâce à la clause c_2 au niveau 5. Nous supposons que le dernier conflit apparu juste avant le redémarrage est représenté par ce graphe de conflit.

L'analyse de conflit qui est l'application de la règle de résolution à partir de la clause conflictuelle en utilisant les différentes implications implicitement encodées dans le graphe d'implications nous permet de visiter le nœud x_{11} qui est le dernier UIP. Au cours de cette analyse, l'ensemble des différents nœuds visités

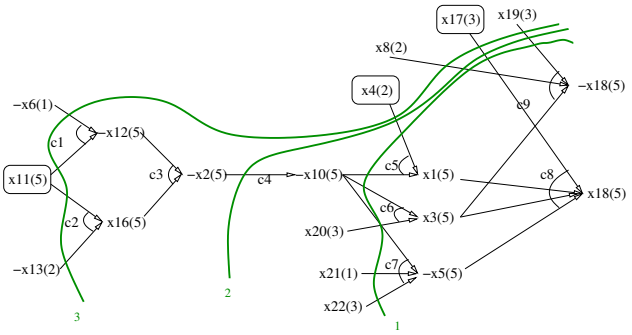


FIGURE 1 – Implication Graph $\mathcal{G}^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$

est $\{ x_{18}, \neg x_5, x_3, x_1, \neg x_{10}, \neg x_2, \neg x_{12}, x_{16}$ et $x_{11}\}$. Cet ensemble de variables est collecté dans la queue en suivant l'ordre selon lequel les différents nœuds sont visités. Au prochain redémarrage, le solveur se branche en priorité sur cet ensemble de variables.

3 Expérimentations

Nos tests furent effectués sur un cluster Intel Xeon quadcore avec 32GB de RAM à 2.66 Ghz. Pour chaque instance, nous utilisons un temps limite d'une heure. Nous utilisons un ensemble d'instances industrielles pris des compétitions SAT 2009 et 2011. Le nombre d'instances différentes correspond à 559.

MiniSAT avec notre stratégie ($MiniSat$ + $Intensification$) résoud 12 instances de plus que MiniSAT sans intensification. Le nombre d'instances résolues en plus est clairement significatif dans la résolution pratique de SAT. En effet, si on observe la récente compétition SAT 2011, le solveur classé premier résoud seulement 4 instances de plus que le solveur classé deuxième. Ces résultats obtenus par notre simple recherche basée sur l'intensification sont représentés dans la figure 2 et figure 3. Ils représentent les résultats des temps cumulés i.e le nombre d'instances (axe-x) résolues en un temps donné en secondes (axe-y). La figure 2 (respectivement figure 3), montre les résultats obtenus sur les instances satisifiables (respectivement insatisfiables). Les améliorations sont souvent plus importantes sur les instances insatisfiables.

Table 1 montre les résultats sur certaines familles d'instances SAT. Cette table montre des améliorations consistantes presque sur toutes les instances quand notre intensification est intégré à MiniSAT. Nous observons une croissance du nombre d'instances résolues. Par exemple, si nous considérons la famille goldb-heqc, (MiniSAT + Intensification) résoud 2 instances de plus que MiniSAT (les instances goldb-heqc-frg1mul et goldb-heqc-x1mul). Globalement, les améliorations

sont d'un ordre de magnitude (les instances velev-vliw-uns-4.0-9C1, 9dlx_vliw_at_b_iq1 et velev-pipe-sat-1.0-b10). Ces derniers résultats démontrent clairement que sur certaines familles, MiniSAT avec Intensification améliore clairement la version basique de MiniSAT. La table 1 confirme aussi que ces améliorations sont plus significatives sur les instances SAT insatisfiables.

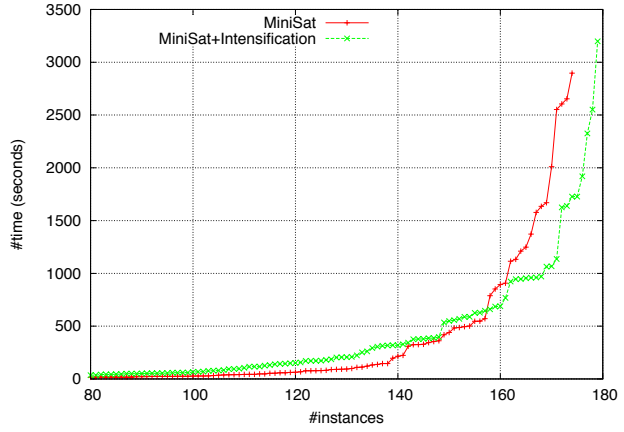


FIGURE 2 – Résultats sur les compétitions SAT 2009-2011 - instances Satisfiables - (Catégorie Industrielles)

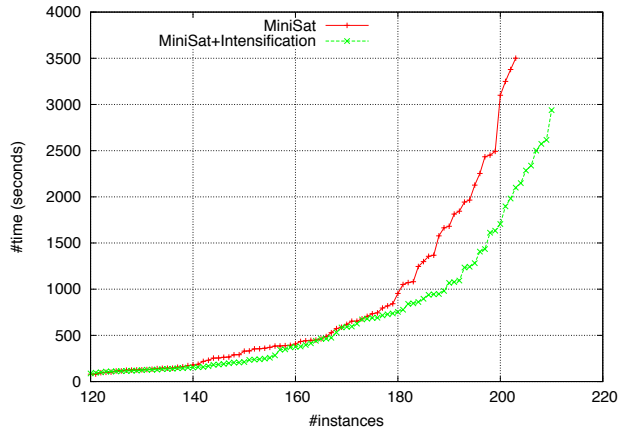


FIGURE 3 – Résultats sur les compétitions SAT 2009-2011 - instances Insatisfiables - (Catégorie Industrielles)

4 Remerciements

Ce travail a été soutenu par l'Agence Nationale de la Recherche - projet ANR programme blanc TUPLES.

Instance	SAT ?	MiniSat	MiniSat+I
goldb-heqc-alu4mul	N	140.94	128.64
goldb-heqc-term1mul	N	54.79	38.40
goldb-heqc-i10mul	N	187.06	129.14
goldb-heqc-dalumul	N	1663.95	181.31
goldb-heqc-frg1mul	N	–	715.66
goldb-heqc-x1mul	N	–	2500.59
velev-engi-uns-1.0-4nd	N	9.12	14.60
velev-live-uns-2.0-ebuf	N	18.53	9.84
velev-pipe-sat-1.0-b7	Y	21.51	23.51
velev-vliw-uns-4.0-9C1	N	3378.39	112.87
velev-pipe-o-uns-1.1-6	N	619.57	28.26
velev-pipe-o-uns-1.0-7	N	446.30	441.81
9dlx_vliw_at_b_iq1	N	1964.90	28.86
9dlx_vliw_at_b_iq2	N	–	70.07
9dlx_vliw_at_b_iq7	N	2642.42	1282.10
9dlx_vliw_at_b_iq3	N	–	189.83
9dlx_vliw_at_b_iq4	N	1750.56	283.74
9dlx_vliw_at_b_iq8	N	2293.63	1633.04
9dlx_vliw_at_b_iq9	N	2410.76	2572.97
9dlx_vliw_at_b_iq5	N	1631.90	464.47
9dlx_vliw_at_b_iq6	N	1267.90	840.68
velev-pipe-uns-1.0-8	N	–	414.21
velev-vliw-uns-4.0-9-i1	N	2095.14	592.52
velev-pipe-sat-1.0-b10	N	77.69	4.61

TABLE 1 – Résultats sur certaines familles d'instances industrielles

5 conclusion

Dans ce papier, nous proposons une nouvelle approche de recherche basée sur l'intensification. Cette recherche basée sur l'intensification est appliquée à chaque redémarrage du solveur SAT. Il collecte les variables rencontrées durant la dernière analyse de conflit. En utilisant un ordre prédéfini. Ces variables sont encore selectionnées jusqu'au sommet de l'arbre de recherche pendant le prochain redémarrage. Ce simple principe d'intensification donne des améliorations significatives quand il est intégré aux solveurs SAT état de l'art.

Notre motivation à l'origine de ce papier est de montrer qu'il reste encore des possibilitées l'améliorations des heuristiques d'ordonnancement des variables SAT. A notre connaissance, cette direction n'a pas retenue beaucoup d'attentions. Les améliorations obtenues par notre simple stratégie d'intensification suggère que des études plus poussées sur le lien entre les redémarrages, les heuristiques d'ordonnancement des variables et l'apprentissage sont nécessaires.

Références

[1] Armin Biere. Adaptive restart strategies for conflict driven sat solvers. In *International Conference,Theory and Applications of Satisfiability Testing, SAT'2008*, pages 28–33, 2008.

[2] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397, 1962.

[3] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International Conference,Theory and Applications of Satisfiability TestingSAT'2003*, pages 502–518, 2003.

[4] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the National Conference on Artificial Intelligence, AAAI'98*, pages 431–437, Madison, Wisconsin, 1998.

[5] Long Guo, Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. Diversification and intensification in parallel sat solving. In *International Conference on Principles and Practice of Constraint Programming, CP'2010*, pages 252–265, 2010.

[6] Y. Hamadi, S. Jabbour, and L. Sais. ManySAT : a parallel SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6 :245–262, 2009.

[7] Jinbo Huang. The effect of restarts on the efficiency of clause learning. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'2007*, pages 2318–2323, 2007.

[8] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic restart policies. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'02)*, pages 674–682, 2002.

[9] Joao P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, November 1996.

[10] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *Design Automation Conference, DAC'01*, pages 530–535, 2001.

[11] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *International Conference,Theory and Applications of Satisfiability Testing, SAT'2007*, pages 294–299, 2007.

[12] Knot Pipatsrisawat and Adnan Darwiche. Width-based restart policies for clause-learning satisfiability solvers. In *International Conference,Theory and Applications of Satisfiability Testing, SAT'2009*, pages 341–355, June 2009.

[13] R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'2003*, pages 1173–1178, 2003.

[14] R. Williams, C. Gomes, and B. Selman. On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In *International Conference on Theory and Applications of Satisfiability Testing, SAT'2003*, pages 222–230, 2003.

Sur la complexité des algorithmes de backtracking et quelques nouvelles classes polynomiales pour CSP *

Achref El Mouelhi[1] **Philippe Jégou**[1] **Cyril Terrioux**[1] **Bruno Zanuttini**[2]

[1] LSIS - UMR CNRS 7296, Aix-Marseille Université
Avenue Escadrille Normandie-Niemen, 13397 Marseille Cedex 20
[2] GREYC, Université de Caen Basse-Normandie, CNRS UMR 6072, ENSICAEN
Campus II, Boulevard du Maréchal Juin, 14032 Caen Cedex
{achref.elmouelhi, philippe.jegou, cyril.terrioux}@lsis.org
bruno.zanuttini@unicaen.fr

Résumé

L'étude des classes polynomiales, pour les problèmes de satisfaction de contraintes (CSP), constitue depuis longtemps un domaine de recherche important qui s'avère aujourd'hui très actif. Cependant, les travaux réalisés jusqu'à présent se sont révélés pour l'essentiel théoriques. En effet, ils se cantonnent en général à la définition de classes d'instances pour lesquelles des algorithmes polynomiaux ad hoc, à la fois pour la reconnaissance et pour la résolution, sont proposés. Ces algorithmes ne peuvent être, en fait, utilisés que pour le traitement d'une classe d'instances donnée. Ils s'avèrent ainsi difficilement exploitables en pratique, et ne sont donc pas exploités au sein de solveurs généraux. L'intérêt pratique des classes polynomiales est ainsi très limitée.

Dans cet article, nous abordons la question des classes polynomiales CSP d'un point de vue différent de l'approche classique, en nous intéressant aux algorithmes que l'on peut retrouver dans les systèmes de résolution opérationnels. Pour cela, nous étudions d'abord la complexité d'algorithmes génériques de résolution de CSP tels que le Forward-Checking par exemple. Cette étude s'appuie sur l'exploitation d'un paramètre issu de la théorie des graphes, et qui permet de proposer de nouvelles bornes de complexité. La mise en relation de ces nouvelles bornes avec certains résultats issus de la théorie des graphes nous permet d'exhiber de nouvelles classes polynomiales. De cette façon, nous montrons comment des algorithmes classiques de résolution de CSP peuvent traiter efficacement en pratique ainsi qu'en théorie, des instances de CSP, sans devoir reconnaître au préalable leur appartenance à d'éventuelles classes polynomiales.

*Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet TUPLES (ANR-2010-BLAN-0210).

Abstract

The question of tractable classes of constraint satisfaction problems (CSPs) has been studied for a long time, and is now a very active research domain. However, studies of tractable classes are typically very theoretical. They usually introduce classes of instances together with polynomial time algorithms for recognizing and solving them, and the algorithms can be used only for the new class.

In this paper, we address the issue of tractable classes of CSPs from a different perspective. We investigate the complexity of classical, generic algorithms for solving CSPs (such as Forward Checking). We introduce a new parameter for measuring their complexity and derive new complexity bounds. By relating the complexity of CSP algorithms to graph-theoretic parameters, our analysis allows us to point at new tractable classes, which can be solved directly by the usual CSP algorithms in polynomial time, and without the need to recognize the classes in advance.

1 Introduction

Les Problèmes de Satisfaction de Contraintes (CSP, [18]) constituent un formalisme important de l'Intelligence Artificielle pour exprimer et résoudre efficacement un large éventail de problèmes réels. Un réseau de contraintes (ou CSP) est constitué d'un ensemble

de variables X, dont chacune doit être affectée à une valeur issue de son domaine (fini) associé, de sorte que ces affectations satisfassent tout un ensemble fini C de contraintes.

Décider si un CSP donné possède une solution est un problème NP-complet. Aussi, les approches classiques proposées pour la résolution de ce problème sont basées sur des algorithmes de recherche, dont la complexité temporelle, dans le pire des cas est de l'ordre de $O(min(n, e).d^n)$ où n est le nombre de variables, e le nombre de contraintes et d la taille du plus grand domaine. L'amélioration de l'efficacité de ce type d'algorithmes s'appuie généralement sur des techniques de filtrages opérés lors de la recherche (en plus d'autres techniques telles que les heuristiques de choix de variables). Avec l'aide de ces techniques, en dépit de leur complexité théorique exponentielle, des algorithmes tels que Forward-Checking [11] (noté FC), RFL (pour Real Full Look-ahead, [14]) ou MAC (pour "Maintaining Arc-Consistency", [19]) pour les CSP binaires, ou nFC$_i$ pour le cas de versions non-binaires [1] s'avèrent très efficaces dans la pratique sur un large éventail de problèmes réels.

Dans une direction orthogonale, d'autres travaux portent sur l'efficacité de la résolutions des CSP en définissant des *classes polynomiales*. Une classe polynomiale est un sous-ensemble d'instances de CSP qui peuvent être reconnues, puis résolues en utilisant des algorithmes de complexité polynomiale. Différents types de classes polynomiales ont été introduites. Certaines d'entre elles sont basées sur la *structure* du réseau de contraintes, par exemple l'acyclicité du réseau [8] ou plus généralement, les réseaux dont la largeur arborescente est bornée par une constante [10]. D'autres classes sont basées sur des restrictions de contraintes (on parle alors de *langage de contraintes*). Par exemple, les contraintes Zero-One-All (ZOA) restreignent les relations de compatibilité à certaines formes [4]. Plus récemment, des classes dites *hybrides* ont été proposées. On peut citer notamment la classe des instances vérifiant la Broken-Triangle Property [5].

Malheureusement, la plupart des classes polynomiales se présentent rarement en pratique, ce qui diminue d'autant leur intérêt. En revanche, comme évoqué ci-dessus, des algorithmes tels que FC, RFL, MAC, ou nFC$_i$, dont la complexité théorique est exponentielle, sont à la base de systèmes pratiques pour la résolution de contraintes, et leurs résultats pratiques sont souvent impressionnants en termes de temps de calcul, alors même qu'ils n'exploitent *a priori* aucune classe polynomiale.

Dans cet article, nous essayons d'amoindrir le fossé existant entre les travaux théoriques sur les classes polynomiales et l'efficacité pratique des méthodes usuelles, cela en tentant de fournir des éléments de réponse à la question portant sur les raisons de l'efficacité observée pour des algorithmes tels que (n)FC ou (n)RFL. Nous le faisons en réévaluant leur complexité en temps en utilisant un nouveau paramètre, à savoir le nombre de cliques maximales dans la *micro-structure* [12] d'une instance ou dans la *micro-structure généralisée* pour le cas non-binaire. Pour le cas binaire, si $\omega_{\#}(\mu(P))$ exprime le nombre de cliques maximales figurant dans la micro-structure d'un CSP P, nous montrons que la complexité d'un algorithme tel que FC est en $O(n^2 d \cdot \omega_{\#}(\mu(P)))$. Cela fournit une nouvelle perspective pour l'étude de l'efficacité des algorithmes de type backtracking en l'associant à un paramètre bien connu en théorie des graphes. En particulier, en réutilisant des résultats connus de la théorie des graphes, nous proposons de nouvelles classes polynomiales de CSP. Le trait saillant de ces classes est qu'elles sont résolues en temps polynomial par des algorithmes à la fois *très généraux* et *largement utilisés*, *sans requérir la nécessité de disposer d'algorithmes de reconnaissance de classes*. À cet égard, notre étude est très proche dans l'esprit de l'étude menée par Rauzy dans le cadre la satisfiabilité de formules propositionnelles et le comportement de l'algorithme DPLL sur les cas connus de classes polynomiales SAT [15].

Cet article est organisé comme suit. Nous présentons d'abord les notations classiques de CSP et les définitions et propriétés de base de la micro-structure. Ensuite, nous présentons notre analyse de la complexité de BT, FC, et RFL sur les CSPs binaires, puis nous introduisons la notion de *micro-structure généralisée* de sorte à étendre notre étude aux CSP non-binaires et donc à des algorithmes de la classe nFC$_i$. Nous mettons ensuite en évidence de nouvelles classes polynomiales issues de la théorie des graphes, qui peuvent ainsi être exploitées dans le domaine des CSP. Enfin, nous évoquons les perspectives offertes pour l'extension de ce travail qui nous apparaît à ce jour comme préliminaire

2 Preliminaries

2.1 Notions de base

Avant d'examiner l'analyse classique de la complexité des algorithmes usuels, nous rappelons quelques notions de base sur les CSP et leur micro-structure.

Définition 1 (CSP) *Un* problème de satisfaction de contraintes *fini (CSP) est un triplet* (X, D, C)*, où* $X = \{x_1, \ldots, x_n\}$ *est un ensemble de variables,* $D = \{D(x_1), \ldots, D(x_n)\}$ *est un ensemble de domaines finis de valeurs, un pour chaque variable, et*

$C = \{c_1, \ldots, c_e\}$ *est un ensemble fini de contraintes. Chaque contrainte* c_i *est un couple* $(S(c_i), R(c_i))$*, où* $S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ *est la* portée *ou* scope *de* c_i*, et* $R(c_i) \subseteq d(x_{i_1}) \times \cdots \times d(x_{i_k})$ *est sa* relation de compatibilité*. L'*arité *de* c_i *est* $|S(c_i)|$.

Une contrainte binaire de portée $\{x_i, x_j\}$ sera notée c_{ij}. Un *CSP binaire* est un CSP pour lequel toutes les contraintes sont binaires. Sinon (cas général), le CSP est dit *n-aire*.

Nous supposons que toutes les variables figurent dans la portée d'au moins une contrainte et que pour une portée donnée, il existe au plus une contrainte. Cela ne pose pas de problème dans le contexte de ce travail puisque deux contraintes portant sur le même ensemble de variables peuvent être fusionnées en une seule en prenant l'intersection de leurs relations.

Définition 2 (affectation, solution) *Étant donné un CSP* (X, D, C)*, une* affectation *de valeurs à* $Y \subseteq X$ *est un ensemble de paires* $t = \{(x_i, v_i) \mid x_i \in Y\}$ *(que l'on écrira* $t = (v_1, \ldots, v_k)$ *quand aucune confusion ne se présente), avec* $v_i \in D(x_i)$ *pour tout* i*. Une affectation de* $Y \subseteq X$ *est dite* cohérente *(ou* consistante *ou bien encore* solution partielle*) si toutes les contraintes* $c \in C$ *dont la portée* $S(c) \subseteq Y$ *sont satisfaites, i.e.,* $t[S(c)] \in R(c)$ *où* $t[S(c)]$ *est la restriction de* t *à* $S(c)$*. Une* solution *est une affectation cohérente de* X.

Notation 1 (paramètres) *Afin d'exprimer la complexité des algorithmes, nous utiliserons les notations suivantes :*

- n *exprime le nombre de variables d'un CSP*
- d *est la cardinalité du plus grand domaine*
- e *est le nombre de contraintes*
- a *est l'arité maximale pour toutes les contraintes*
- r *est le nombre de tuples de la plus grande relation*

Étant donné un CSP, la question fondamentale est de décider s'il possède une solution, problème bien connu comme étant NP-complet. Afin d'étudier les CSP et essayer de contourner cette difficulté, différents points de vue peuvent être adoptés. Concernant les *CSP binaires*, l'un d'eux s'appuie sur la notion de *micro-structure d'une instance*, qui correspond à son graphe de compatibilité et dont nous rappelons la définition. Intuitivement, les sommets de ce graphe codent les valeurs et ses arêtes codent leur compatibilité.

Définition 3 (micro-structure) *Étant donné un CSP* $P = (X, D, C)$*, la* micro-structure *de* P *est un graphe non orienté* $\mu(P) = (V, E)$ *avec :*

- $V = \{(x_i, v_i) : x_i \in X, v_i \in D(x_i)\}$,
- $E = \{\ \{(x_i, v_i), (x_j, v_j)\}\ \mid\ i \neq j, c_{ij} \notin C$ *ou* $c_{ij} \in C, (v_i, v_j) \in R(c_{ij})\}$

En d'autres termes, la micro-structure d'un CSP binaire P contient une arête pour toutes les paires de sommets, sauf pour les sommets issus d'un même domaine et pour les sommets correspondant à des paires qui sont interdites par certaines contraintes. On peut constater que la micro-structure d'un CSP est un graphe n-parti, car il n'existe pas d'arête reliant les sommets issus d'un même domaine. Dans cet article, nous allons étudier la complexité des algorithmes de résolution de CSP classiques en s'appuyant sur les cliques qui figurent dans la micro-structure.

Définition 4 (clique) *Un* graphe complet *est un graphe simple dans lequel chaque paire de sommets distincts est reliée par une arête. Une* k-clique *dans un graphe non orienté est un sous-ensemble de* k *sommets induisant un graphe complet (tous les sommets sont deux à deux adjacents). Une* clique maximale *est une clique qui n'est pas un sous-ensemble propre d'une autre clique. Nous noterons par* $\omega_{\#}(G)$ *le nombre de cliques maximales dans un graphe* G.

Le résultat suivant se déduit directement du fait que dans une micro-structure, les sommets d'une clique correspondent à des valeurs compatibles issues de domaines différents.

Proposition 1 *Étant donné un CSP binaire* P *et sa micro-structure* $\mu(P)$*, une affectation* $(v_1, ..., v_n)$ *de* X *est une solution de* P *ssi* $\{(x_1, v_1), ..., (x_n, v_n)\}$ *est une* n*-clique de* $\mu(P)$.

On peut facilement constater que la transformation d'un CSP P en sa micro-structure $\mu(P)$ peut être réalisée en temps polynomial. On en déduit directement l'existence d'une réduction polynomiale du problème de décision lié à l'existence de solution d'un CSP binaire donné vers le problème d'existence d'une clique de taille donnée dans un graphe non-orienté (appelé généralement *problème de la clique*). Cette transformation qui offre un point de vue nouveau pour l'étude des CSP en fournissant un lien avec des notions issues de la Théorie des Graphes a d'abord été exploitée par [12] qui a proposé des classes polynomiales de CSPs basées sur l'existence de classes polynomiales pour le problème de la clique (*graphes chordaux* [9]). Une approche identique a été considérée pour le cas des classes polynomiales *hybrides* [3].

2.2 Complexité des Algorithmes de Backtracking

Nous passons maintenant brièvement en revue la complexité des algorithmes qui nous intéressent ici : BT, FC, RFL pour les CSP binaires, et nFC$_i$ pour le cas non-binaire. Ces algorithmes couvrent toutes les approches qui utilisent le backtracking et les approches

de type prospectif au sens lookahead (la question des choix d'ordres de variables et de valeurs ne sera pas considérée ici).

L'algorithme de *Backtracking* (noté BT et également appelé *Backtracking chronologique*) est une procédure d'énumération récursive. Il commence par une affectation vide et dans le cas général, étant donnée une solution partielle courante $(v_1, v_2, \ldots, v_i)$, il choisit une nouvelle variable x_{i+1} et cherche à affecter des valeurs de $D(x_{i+1})$ à x_{i+1}. Le seul contrôle effectué consiste alors à vérifier que l'affectation résultante $(v_1, v_2, \ldots, v_i, v_{i+1})$ est cohérente. Dans l'affirmative, BT continue avec cette nouvelle solution partielle en l'étendant à une nouvelle variable non encore affectée (appelée *variable future*). Sinon (si $(v_1, v_2, \ldots, v_i, v_{i+1})$ n'est pas compatible), BT essaie une autre valeur de $D(x_{i+1})$. S'il n'y a plus de valeur inexplorée, BT se trouve dans une impasse, et il *désinstancie* x_i (il effectue un *backtrack*).

Il est facile de voir que la recherche effectuée par BT correspond à un parcours en profondeur d'abord d'un arbre sémantique appelé *arbre de recherche*, et dont la racine est un tuple vide, tandis que les nœuds situés au $i^{ème}$ niveau sont des i-uplets qui représentent les affectations des variables le long du chemin correspondant dans l'arbre. Les nœuds de cet arbre qui correspondent à des solutions partielles sont appelés *nœuds consistants*, tandis que les autres nœuds sont appelés *nœuds inconsistants*. Le nombre de nœuds dans l'arbre de recherche est au plus $\Sigma_{0 \leq i \leq n} d^i = \frac{d^{n+1}-1}{d-1}$, par conséquent, il est en $O(d^n)$. Ainsi, la complexité de BT est bornée par le nombre de nœuds multiplié par le coût à chaque nœud. En supposant qu'un test de contrainte peut être réalisé en $O(a)$, la complexité de BT est de $O(min(n, e)ad^n)$.

BT peut être considéré comme un algorithme générique. Des algorithmes basés sur BT et utilisés en pratique réalisent un certain travail supplémentaire à chaque nœud de l'arbre de recherche, à savoir, ils suppriment des valeurs incompatibles dans le domaine des variables futures (procédé appelé *filtrage*). Dans le cas des CSP *binaires*, FC supprime les valeurs incompatibles avec la dernière affectation réalisée, tandis que RFL applique la consistance d'arc (AC) sur les variables futures. La complexité de FC peut être bornée par $O(nd^n)$. En utilisant un algorithme en $O(ed^2)$ pour réaliser AC, la complexité de la RFL est en $O(ed^2d^{n-1}) = O(ed^{n+1})$. Dans le cas des CSP *n-aires*, les algorithmes de la classe nFC_i $(i = 0, 1 \ldots 5)$ recouvrent l'application partielle ou totale de *la consistance d'arc généralisée* (GAC) sur un sous-ensemble de contraintes impliquant à la fois les variables affectées et les variables futures. Dans chaque cas, le filtrage est réalisé après chaque affectation de variable. Ainsi, la complexité des techniques de type nFC_i dépend du coût du filtrage. Pour nFC_5 qui réalise le filtrage le plus puissant, la complexité est alors en $O(n^e ard)$. C'est la mème chose si l'on considère la version n-aire de RFL qui maintient GAC à chaque nœud. Dans ce qui suit, nous noterons par nBT et nRFL les versions n-aires de BT et de RFL [1].

Nous tenons à souligner ici que les algorithmes (n)BT, FC, (n)RLF ou nFC_i peuvent utiliser un ordre *dynamique* sur les variables, c'est-à-dire que le choix de la future variable (x_{i+1}) à explorer peut être décidé après chaque nouvelle affectation. Notons cependant qu'ici, lorsque l'affectation de la valeur v_{i+1} conduit à une impasse, nous ne considérons pas la possibilité de remplacer x_{i+1} par une autre variable future (quand bien même il resterait encore des valeurs à explorer pour x_{i+1}) comme le fait par exemple MAC. Au contraire, nous nous en tenons ici aux algorithmes qui changent la variable courante seulement après avoir épuisé son domaine.

3 Une Nouvelle Analyse de la Complexité pour les CSP binaires

Nous arrivons maintenant au cœur de notre contribution, à savoir une analyse de la complexité des algorithmes classiques en termes de paramètres liés à la micro-structure. Dans ce qui suit, nous disons qu'un nœud de l'arbre de recherche est un *nœud consistant maximalement profond* s'il est cohérent et s'il ne possède pas de nœud fils cohérent (sur la variable suivante dans l'ordre). Ainsi, un tel nœud correspond soit à une solution, soit à une solution partielle qui ne peut être étendue de façon cohérente sur la variable suivante.

Le résultat suivant peut-être considéré comme central pour notre étude.

Proposition 2 *Étant donné un CSP binaire $P = (X, D, C)$, il existe une application injective de l'ensemble des nœuds consistants maximalement profonds explorés par BT vers l'ensemble des cliques maximales de $\mu(P)$.*

Preuve : Soit $(v_1, v_2, \ldots, v_i)$ un nœud consistant maximalement profond exploré par BT. Par définition, $(v_1, v_2, \ldots, v_i)$ est une solution partielle, et donc pour tout $1 \leq j, k \leq i$, soit il n'existe pas de contrainte c_{jk} dans C dont la portée est $\{x_j, x_k\}$, soit (v_j, v_k) figure dans la relation $R(c_{jk})$. Dans les deux cas, $\{(x_j, v_j), (x_k, v_k)\}$ constitue une arête de $\mu(P)$. Donc $\{(x_1, v_1), \ldots, (x_i, v_i)\}$ est une clique de $\mu(P)$ et donc,

1. Notons que nBT correspond exactement au même algorithme que BT, ce qui n'est pas le cas de nRFL puisqu'il faut alors considèrer la consistance d'arc généralisée GAC

elle est incluse dans au moins une clique maximale de $\mu(P)$. Nous la noterons $Cl(v_1, v_2, \ldots, v_i)$.

Nous montrons maintenant que Cl constitue une application injective de l'ensemble des nœuds consistants maximalement profonds vers l'ensemble des cliques maximales. Par construction de BT, si $(v_1, v_2, \ldots, v_i)$ et $(v'_1, v'_2, \ldots, v'_{i'})$ sont deux nœuds maximalement profonds explorés, ils doivent différer d'au moins une valeur sur une variable. Précisément, ils doivent différer au moins sur le nœud où les chemins correspondants se diffèrencient dans l'arbre de recherche, et donc sur les nœuds correspondant à une certaine variable x_j affectée à une certaine valeur sur un chemin, et à une autre valeur sur l'autre[2]. Comme il n'y a pas d"arête de $\mu(P)$ connectant deux valeurs d'une même variable, il ne peut pas y avoir de clique maximale contenant à la fois $(v_1, v_2, \ldots, v_i)$ et $(v'_1, v'_2, \ldots, v'_{i'})$, donc Cl est nécessairement une application injective. □

En utilisant cette propriété, nous pouvons facilement borner le nombre de nœuds figurant dans un arbre de recherche induit par une recherche de type backtracking, et donc aussi sa complexité en temps, en termes de propriété liée à sa micro-structure. Comme il est d'usage, nous supposons qu'un test de contrainte (vérifier si $(v_i, v_j) \in R(c_{ij})$) est réalisable en temps constant.

Proposition 3 *Le nombre de nœuds $N_{BT}(P)$ figurant dans l'arbre de recherche développé par BT pour résoudre un CSP binaire $P = (X, D, C)$, vérifie $N_{BT}(P) \leq nd \cdot \omega_{\#}(\mu(P))$. Sa complexité en temps est en $O(n^2 d \cdot \omega_{\#}(\mu(P)))$.*

Preuve : Considérons d'abord le nombre de nœuds consistants. Parce que n'importe quel nœud de l'arbre de recherche figure à une profondeur au plus égale à n et que le chemin de la racine à un nœud consistant contient uniquement des nœuds consistants, comme corollaire direct de la Proposition 2, nous obtenons que l'arbre de recherche contient au plus $n \cdot \omega_{\#}(\mu(P))$ nœuds consistants. Maintenant, par définition de BT, un nœud consistant possède au plus d fils (un par valeur candidate pour la variable suivante), et les nœuds incompatibles n'en ont pas. Il s'ensuit que l'arbre de recherche a au plus $nd \cdot \omega_{\#}(\mu(P))$ nœuds de toute nature.

La complexité en temps se déduit directement puisque chaque nœud correspond à l'extension de l'affectation partielle à une variable supplémentaire x_{i+1}, ce qui implique au plus un test de contrainte par autre variable déjà affectée (vérifier la satisfaction de $c_{j(i+1)}$ pour chaque x_j déjà affecté), et comme il n'en existe nécessairement moins de n, on obtient le résultat.□

On peut constater dans l'énoncé de la Proposition 3 ainsi que dans les suivantes, que le nombre de cliques maximales $\omega_{\#}(\mu(P))$ pourrait être remplacé par le nombre de cliques maximales *de taille au plus* $n-1$. Ceci parce que dès qu'un chemin exploré est contenu dans une n-clique, c'est-à-dire, dans une solution, aucun retour arrière ne se produira ultérieurement sur ce chemin.

Nous analysons maintenant FC et RFL. Il apparaît clairement que la Proposition 2 vaut également pour chacun de ces algorithmes. Le nombre de nœuds explorés découle du fait que seuls les nœuds consistants sont explorés. La complexité en temps se déduit du fait qu'au plus n domaines futurs sont filtrés par FC et que AC est réalisable en un temps $O(ed^2)$ par RFL.

Proposition 4 *Le nombre de nœuds $N_{FC}(P)$ figurant dans l'arbre de recherche développé par FC ou par RFL pour résoudre un CSP binaire $P = (X, D, C)$, vérifie $N_{FC}(P) \leq n \cdot \omega_{\#}(\mu(P))$. La complexité en temps de FC est en $O(n^2 d \cdot \omega_{\#}(\mu(P)))$, et celle de RFL est en $O(ned^2 \cdot \omega_{\#}(\mu(P)))$.*

Il est important de constater que la taille relative des arbres de recherche de BT et FC fournie par l'analyse classique et celle obtenue par l'approche que nous proposons sont les mêmes. Notamment, dans le pire des cas, l'arbre de recherche de BT est d fois plus grand en nombre de nœuds que celui de FC sur la même instance.

4 Une Nouvelle Analyse pour les CSP Non-Binaires

4.1 Micro-structure Généralisée

Nous passons maintenant à l'étude des CSP n-aires. Dans un premier temps, nous étendons la notion de micro-structure à ce cas général, puis nous analysons la complexité des techniques de type nFC$_i$ en termes de nombre de cliques maximales.

Notons que la généralisation de la notion de la micro-structure a d'abord été proposée dans [3]. Toutefois, cette notion est basée sur les hypergraphes et n'a pas vraiment été exploitée jusqu'à présent. En revanche, notre notion se situe toujours dans le cadre des graphes. Notre *micro-structure généralisée* est obtenue en considérant comme sommets les tuples figurant dans les relations du CSP plutôt que les valeurs (x_i, v_i) utilisées dans le cas binaire.

2. Nous utilisons ici l'hypothèse selon laquelle l'algorithme explore toutes les valeurs d'une variable avant le réordonnancement des variables futures

Définition 5 (micro-structure généralisée) *Étant donné un CSP $P = (X, D, C)$ (pas nécessairement binaire), la* micro-structure généralisée *de P est un graphe non-orienté $\mu_G(P) = (V, E)$ avec :*

- $V = \{(c_i, t_i) : c_i \in C, t_i \in R(c_i)\}$,
- $E = \{\ \{(c_i, t_i), (c_j, t_j)\}\ \mid\ i \neq j, t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]\}$

Comme pour la micro-structure des CSP binaires, il existe une relation directe entre cliques et solutions de CSP :

Proposition 5 *Un CSP P possède une solution ssi $\mu_G(P)$ possède une clique de taille e.*

Preuve : Par construction, $\mu_G(P)$ est e-parti et chaque clique contient au plus un sommet (c_i, t_i) par contrainte $c_i \in C$. Donc, les e-cliques de $\mu_G(P)$ correspondent exactement à des cliques ne possédant qu'un sommet (c_i, t_i) par contrainte $c_i \in C$. De plus, et encore par construction de $\mu_G(P)$, tout couple de sommets (c_i, t_i), (c_j, t_j) joints par une arête vérifie $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$. Donc tous les t_i d'une clique sont connectés ensemble, et il s'ensuit que les e-cliques de $\mu_G(P)$ correspondent exactement à des ensembles de tuples t qui sont connectés avec d'autres tuples permis par les contraintes, c'est-à-dire qu'ils correspondent à des solutions de P. □

On peut observer que la généralisation de la micro-structure correspond à la micro-structure de la *représentation duale* d'un CSP [6]. À ce titre, on peut aussi constater que notre généralisation correspond en fait au *line-graph* ou *graphe des intersections* (c'est le graphe dual) de l'hypergraphe proposé dans [3], auquel nous aurions rajouté des arêtes pour le cas où deux portées de contraintes ne s'intersecteraient pas. Enfin, et dans le même esprit que celui de notre approche, nous pourrions proposer d'autres généralisations de micro-structure en considérant toute représentation graphique de CSP n-aire, dès que la représentation a le même ensemble de solutions - à une bijection près - que l'instance d'origine (par exemple le *codage par variables cachées* [17]). Toutefois, par manque de place, nous ne traitons pas ces questions ici.

4.2 Complexité de nBT et nFC

Nous étudions maintenant la complexité des algorithmes de résolution des CSP n-aires. Dans un premier temps, nous posons une hypothèse sur l'ordre dans lequel ces algorithmes explorent les variables, puis nous discutons de cette restriction.

Définition 6 (ordre compatible) *Soit P un CSP. Un ordre total $(x_1, x_2, \ldots x_n)$ sur X est dit* compatible avec les contraintes de C *s'il y a k contraintes $c_{i_1}, c_{i_2}, \ldots c_{i_k}$ dans C ($1 \leq k \leq e$) qui vérifient :*

- $\bigcup_{1 \leq l \leq k} S(c_{i_l}) = X$
- *il y a k variables $x_{i_1}, x_{i_2}, \ldots x_{i_k}$ telles que pour tout ℓ dans $\{1, \ldots, k\}$, $x_\ell \in S(c_{i_\ell})$ et $\bigcup_{1 \leq j \leq \ell} S(c_{i_j}) = \{x_i \mid i = 1, \ldots, i_\ell\}$.*

En d'autres termes, l'ordre est tel que les variables figurant dans la portée d'une première contrainte c_{i_1} apparaissent toutes d'abord, puis toutes les variables figurant dans la portée d'une certaine contrainte C_{i_2} apparaissent ensuite (sauf pour celles qui figurent déjà dans c_{i_1}), etc. Les variables $x_{i_1}, \ldots, x_{i_k}$ dans la définition sont telles que x_{i_j} est la dernière variable affectée dans la portée de c_{i_j}. Nous nous considèrerons ces variables comme étant des *jalons* dans l'ordre.

Par exemple, avec la notation de la définition, nous devons avoir $S(c_{i_1}) = \{x_1, x_2, \ldots x_{i_1}\}$ et $S(c_{i_1}) \cup S(c_{i_2}) = \{x_1, x_2, \ldots x_{i_1}, x_{i_1+1}, \ldots x_{i_2}\}$. La variable x_{i_1} est un jalon (dernière variable affectée dans la portée de c_{i_1}) de même que x_{i_2} pour c_{i_2}.

Dans l'hypothèse où un tel ordre sur les variables est utilisé, nous pouvons donner une généralisation de la Proposition 2.

Proposition 6 *Soit P un CSP n-aire. Supposons que nBT explore les variables dans un ordre compatible avec les contraintes de C. Alors, il existe une application injective de l'ensemble des nœuds consistants maximalement profonds (x_i, v_i) de l'arbre de recherche tels que x_i est un jalon, vers l'ensemble des cliques maximales de $\mu_G(P)$.*

Preuve : Soit t une affectation correspondant à un nœud tel que défini, et notons x_{i_j} pour la dernière variable affectée dans t (x_{i_j} est un jalon par hypothèse). De même, soit t' une autre affectation maximale compatible avec le jalon $x_{i_{j'}}$ comme dernière variable. Notons T l'ensemble $\{t[S(c)] \mid c \in C, S(c) \subseteq \{x_1, \ldots, x_{i_j}\}\}$, c'est-à-dire, l'ensemble de toutes les projections de t sur les portées de toutes les contraintes totalement affectées t, et idem pour T'. Alors T (resp. T') est inclus dans une clique maximale $Cl(t)$ (resp. $Cl(t')$) de $\mu_G(P)$). Maintenant, supposons $j' \geq j$ (sans manque de généralité). Avec $t \neq t'$, le fait que t soit maximalement consistant, et le fait que t' soit consistant, permet d'affirmer que t' diffère de t d'au moins une variable x_ℓ avec $\ell \leq i_j$. Ainsi, cette variable est affectée différemment dans chaque cas, et il existe une certaine contrainte c_{i_ℓ} telle que $t[S(c_{i_\ell})]$ est différent de $t'[S(c_{i_\ell})]$, et donc, t et t' ne peuvent être inclus dans une même clique. Donc, Cl définit une application injective de l'ensemble des affectations considérées vers l'ensembles des cliques maximales de $\mu_G(P)$. □

En utilisant cette propriété, nous pouvons borner le nombre de nœuds dans un arbre de recherche induit

par une recherche de type backtracking, ainsi que sa complexité en temps, relativement à la micro-structure généralisée.

Proposition 7 *Soit $P = (X, D, C)$ un CSP n-aire. Supposons que nBT utilise un ordre sur les variables qui est compatible avec les contraintes de C. Le nombre de nœuds $N_{nBT}(P)$ dans l'arbre de recherche de nBT sur P vérifie $N_{nBT}(P) \leq nd^a \cdot \omega_{\#}(\mu_G(P))$. Sa complexité en temps est en $O(nea \cdot d^a \cdot \omega_{\#}(\mu_G(P)))$.*

Preuve : Du fait de la Proposition 6, le sous-arbre induit par l'arbre de recherche sur les jalons contient au plus $\omega_{\#}(\mu_G(P))$ nœuds. Pour atteindre un jalon à partir du précédent, c'est-à-dire, pour étendre une affectation de $x_1, \ldots, x_{i_j}$ à l'affectation de $x_1, \ldots, x_{i_{j+1}}$ (avec les notations de la définition 6), nBT explore au plus a variables (ceci par définition d'un ordre compatible avec les contraintes). Par conséquent, il explore au plus d^a combinaisons de valeurs (nBT n'a aucune raison pour exclure une affectation avant d'affecter toutes les variables figurant dans la portée d'une contrainte). Puisqu'une branche contient au plus n nœuds, et donc finalement n jalons, on obtient le résultat.

La complexité en temps se déduit directement puisque chaque nœud nécessite au plus e tests de contraintes, chacun en $O(a)$ avec une structure de données appropriée. □

Un résultat similaire est valable pour nFC_i ($i \geq 2$). Néanmoins, nous devons tenir compte du coût supplémentaire dû à l'application de GAC, qui est $O(e \cdot a \cdot r)$ à chaque nœud.

Toutefois, on peut noter que, contrairement à nBT, et en raison de l'utilisation de GAC, nFC_i explore seulement les r tuples autorisés par c lors de l'exploration des variables dans $S(c)$, plutôt que toutes les d^a combinaisons de valeurs.

Proposition 8 *Soit $P = (X, D, C)$ un CSP n-aire. Supposons que nFC_i ($i \geq 2$) utilise un ordre sur les variables qui est compatible avec les contraintes de C. Le nombre de nœuds $N_{nFC_i}(P)$ dans l'arbre de recherche de nFC_i sur P vérifie $N_{nFC_i}(P) \leq nr \cdot \omega_{\#}(\mu_G(P))$. Sa complexité en temps est en $O(nea \cdot r^2 \cdot \omega_{\#}(\mu_G(P)))$.*

Ce résultat est également valable pour nRFL.

4.3 Complexité sans hypothèse sur l'ordre

On pourrait estimer que notre restriction posée sur les ordres d'affectation des variables en termes de compatibilité avec les contraintes n'est pas respectée par tous les ordres raisonnables. Par exemple, il n'y a aucune raison en général, pour qu'une heuristique très usitée telle que *dom / deg* respecte ce type d'ordres. Nous montrons donc ici que ce type de restriction s'avère nécessaire.

Pour montrer cela, nous construisons une famille d'instances qui possèdent un nombre linéaire de cliques dans leur micro-structure généralisée (linéaire en son nombre de sommets), mais pour lesquelles nFC_5 explore un arbre de recherche de taille exponentielle (dans le nombre de sommets) pour un certain ordre sur variable.

Les instances (X, D, C) de cette famille sont construites comme suit. Avec e le nombre de contraintes, nous considérons deux variables distinctes x_0 et x'_0 de X qui seront communes à toutes les contraintes, et nous aurons $X \setminus \{x_0, x'_0\}$ qui est partitionné en e ensembles $X_1, \ldots, X_e$ (X_i sera associé à c_i). Ainsi, chaque contrainte $c_i \in C$ possède une portée $S(c_i) = \{x_0, x'_0\} \cup X_i$. Maintenant, le domaine est $\{v_0, \ldots, v_{e-1}\}$ pour toutes les variables ($d = e$). Enfin, les tuples permis pour une contrainte c_i sont précisément de la forme $\{(x_0, v_j), (x'_0, v_{i+j}), \ldots\}$, pour $j = 1, \ldots, e$ (les indices sont pris modulo e) et sans restriction pour les affectations de X_i. On note que pour $i \neq i'$, les restrictions pour les tuples permis par c_i et $c_{i'}$ sur $\{x_0, x'_0\}$ ne coïncident jamais, de sorte qu'il n'existe aucune arête dans $\mu_G(P)$. Ainsi, le nombre de cliques dans $\mu_G(P)$ exactement égal au nombre de sommets $|V| = e^{2+(n-2)/e}$.

D'autre part, supposons que nFC_5 explore toutes les variables dans les ensembles X_i et explore seulement x_0 et x'_0 après elles. Ainsi, puisque que toutes les valeurs pour x_0 possèdent un support dans toutes les contraintes, et de même pour x'_0, aucune valeur ne sera retirée avant d'atteindre x_0 ou x'_0, et donc toutes les $e^{n-2} \sim |V|^e$ combinaisons de valeurs seront explorées, c'est-à-dire exponentiellement plus que le nombre de cliques figurant dans $\mu_G(P)$.

5 Quelques Classes Classes Polynomiales pour le Backtracking

Le nombre de cliques dans un graphe peut croître de façon exponentielle avec la taille du graphe [20] et il en est de même du nombre $\omega_{\#}(G)$ de *cliques maximales* dans un graphe G [13]. Toutefois, pour certaines classes de graphes, le nombre de cliques maximales peut être borné par un polynôme en la taille du graphe. Si la micro-structure (généralisée) d'un (d'une famille de) CSP P appartient à l'une de ces classes, l'analyse que nous avons présentée dans les sections précédentes permet de conclure que P est résoluble en temps polynomial par des algorithmes classiques d'énumération, et ce, *sans avoir à reconnaître l'appartenance de l'instance à cette classe.*

Dans cette section, nous étudions plusieurs classes

de graphes vérifiant cette propriété et nous commentons leur pertinence en termes de problèmes de satisfaction de contraintes.

5.1 Graphes "Triangle-Free" ou Bipartis

Nous rappelons qu'un *k-cycle* dans un graphe $G = (V, E)$ est une séquence $(v_1, v_2, \dots v_{k+1})$ de sommets distincts, sauf pour $v_1 = v_{k+1}$, vérifiant $\forall i, 1 \leq i \leq k, \{v_i, v_{i+1}\} \in E$.

Un graphe *triangle-free* est un graphe non-orienté sans 3-cycle. Il est facile de voir que le nombre de cliques maximales dans un graphe *triangle-free* est exactement égal au nombre de ses arêtes. En effet, chaque arête est une clique, et par définition, il ne peut pas exister de plus grande clique. Avec notre analyse, nous pouvons affirmer que pour la classe des CSP dont la micro-structure (généralisée) est *triangle-free*, les algorithmes (n)BT, (n)FC et (n)RFL sont des procédures de résolution fonctionnant en temps polynomial. Notons cependant que cette classe de CSP constitue d'une certaine façon un cas *dégénéré*, puisque excepté pour les instances ayant au plus deux variables (cas binaire) ou deux contraintes (cas n-aire), les instances avec micro-structure (généralisée) *triangle-free* sont incohérentes.

Un autre cas dégénéré est constitué par la classe des graphes bipartis. Un graphe est dit *biparti* s'il ne contient pas de cycle impair. Encore une fois, un graphe biparti ne peut contenir de clique de plus de deux sommets, et donc aucune affectation partielle de plus de trois variables ne sera considérée par BT (dont la complexité sera ainsi en $O(d^3)$).

Nous passons maintenant à des classes plus intéressantes, qui aussi, contiennent pour l'essentiel des CSP incohérents, mais pour lesquelles notre analyse fournit une meilleure complexité en temps que celle offerte par l'analyse classique.

5.2 Graphes Planaires, Toroïdaux, et "Embedded"

Définition 7 (planaire) *Un graphe* planaire *est un graphe qui peut être dessiné dans le plan sans que deux arêtes ne se chevauchent.*

[20] a prouvé que le nombre cliques d'un graphe planaire $G = (V, E)$ est au plus $8(|V| - 2)$.

Définition 8 (toroïdal) *Un graphe* toroïdal *est un graphe qui peut être dessiné sur un tore sans que deux arêtes ne se chevauchent.*

[7] ont montré que tout graphe toroïdal possède au plus $8(|V| + 9)$ cliques et que tout graphe intégrable ("Embedded") dans une surface possède un nombre linéaire de cliques puisque majoré par $8(|V| + 27)$ au pire. Puisque la micro-structure $\mu(P)$ (resp. $\mu_G(P)$) d'un CSP P contient nd sommets (resp. er sommets), alors si $\mu(P)$ (resp. $\mu_G(P)$) appartient à l'une de ces classes de graphes, alors $\omega_\#(\mu(P))$ (resp. $\omega_\#(\mu_G(P))$) possède au plus $O(nd)$ cliques (resp. $O(er)$).

Grâce aux Propositions 3-4 et 7-8, nous obtenons immédiatement le résultat suivant.

Théorème 1 *Soit Em qui désigne la classe de tous les CSP dont les micro-structure sont planaires, toroïdales, ou intégrables dans une surface. Alors les instances de Em sont résolues en un temps*

- $O(n^2 d \cdot \omega_\#(\mu(P))) = O(n^3 d^2)$ *par BT or FC,*
- $O(ned^2 \cdot \omega_\#(\mu(P))) = O(n^2 ed^3)$ *par RFL,*
- $O(nead^a \cdot \omega_\#(\mu_G(P))) = O(ne^2 ard^a)$ *par nBT,*
- $O(near^2 \cdot \omega_\#(\mu_G(P))) = O(ne^2 ar^3)$ *par nFC$_i$ ou nRFL.*

Rappelons que cette famille de graphes ne peut contenir en tant que mineur ni une 8-clique (pour les graphes toroïdaux), ou ni une 5-clique ou $K_{3,3}$ (pour les graphes planaires). Cela a pour conséquence, en particulier, que tous les CSP binaires (resp. n-aires) dans *Em* sur au moins 8 variables (resp. contraintes) sont incohérents. On peut donc dire raisonnablement que cette classe est pour le moins dégénérée. Néanmoins, si l'on se réfère à l'analyse classiques de la complexité, on constate que, par exemple, BT résout ces instances en un temps $O(d^8)$, pour le cas où d est grand, ce temps est significativement supérieur à $O(n^3 d^2)$.

5.3 Graphes CSG

Nous étudions maintenant la classe des *Graphes CSG* qui a été introduite par [2] et qui généralise la classe des *graphes chordaux* (dits aussi *triangulés*).

Étant donné un graphe (V, E) et un ordre $v_1, \dots, v_{|V|}$ sur ses sommets, nous noterons $N^+(v_i)$ pour le *voisinage ultérieur* de v_i, c'est-à-dire, $N^+(v_i) = \{v_j \in V | \{v_i, v_j\} \in E, i < j\}$. Pour $V' \subseteq V$, nous notons $G(V')$ le sous-graphe *induit* par E sur V', soit, $G(V') = (V', E')$ où $E' = \{\{x, y\} \mid x, y \in V' \text{ et } \{x, y\} \in E\}$.

Définition 9 (Graphes CSG) *La classe de graphes CSG^k est définie inductivement comme suit.*

- *CSG^0 est la classe des graphes complets.*
- *Étant donné $k > 0$, CSG^k est la classe des graphes $G = (V, E)$ tels qu'il existe un ordre $\sigma = (v_1, ..., v_{|V|})$ sur V vérifiant que pour $i = 1, \dots, |V|$, le graphe $G(N^+(v_i))$ est un graphe CSG^{k-1}.*

La classe de graphes CSG généralise la classe des graphes complets (graphes CSG^0) et surtout la classe

des graphes chordaux (graphes CSG^1). Comme les graphes chordaux, les graphes CSG possèdent des propriétés intéressantes. Par exemple, ils peuvent être reconnus en temps polynomial. De plus il a été démontré dans [2] que les graphes CSG^k possèdent au plus $|V|^k$ cliques maximales, et un algorithme de complexité polynomiale en $O(|V|^{2(k-1)}(|V|+|E|))$ a été proposé pour les énumérer.

L'existence de ces deux algorithmes confère à la classe des CSP qui ont une micro-structure (généralisée) CSG^k le statut de classe polynomiale pour toute valeur k fixée. Nous sommes cependant en mesure de montrer que des algorithmes *génériques* tels que (n)BT, FC, nFC_i ou (n)RFL s'exécutent en temps polynomial sur de tels CSP, sans même qu'il soit nécessaire de reconnaître l'appartenance des instances traitées à cette classe (et donc sans avoir besoin de calculer la micro-structure). À nouveau, ce résultat est issu du nombre de cliques maximales par l'application des Propositions 4 et 7-8.

Théorème 2 *Pour tout entier k fixé, la classe des CSP dont la micro-structure (généralisée) est CSG^k peut être résolue en un temps*

- $O(n^2d \cdot \omega_{\#}(\mu(P))) = O(n^{k+2}d^{k+1})$ *par BT et FC,*
- $O(ned^2 \cdot \omega_{\#}(\mu(P))) = O(n^{k+1}ed^{k+2})$ *par RFL,*
- $O(nead^a \cdot \omega_{\#}(\mu_G(P))) = O(ne^{k+1}ar^kd^a)$ *par nBT,*
- $O(near^2 \cdot \omega_{\#}(\mu_G(P))) = O(ne^{k+1}ar^{k+2})$ *par nFC_i ou nRFL.*

Nous pouvons constater de plus que les complexités temps sont même meilleures que celles issues de l'algorithme dédié. Par exemple, celui-ci calcule la micro-structure d'un CSP binaire et énumère toutes les cliques maximales ou bien s'arrête dès qu'une n-clique est trouvée. Ainsi, la complexité en temps est en $O((nd)^{2(k-1)}(nd+n^2d^2)) = O((nd)^{2k})$. Néanmoins, nous pouvons noter que l'algorithme est défini pour des graphes CSG^k quelconques, alors que les micro-structures (généralisées) de CSP sont des graphes très particuliers.

Il semble, en termes de CSP, que les graphes CSG soient généralement moins restrictifs que les classes précédentes de graphes. Par exemple, il est possible d'avoir des graphes CSG possédant des n-cliques (resp. des e-cliques) pour toute valeur de n (resp. e), contrairement notamment au cas des graphes planaires. En particulier, il existe des CSP cohérents possédant une micro-structure (généralisée) qui est un graphe CSG^k. C'est le cas notamment pour la classe CSG^0 qui sont exactement les CSPs binaires cohérents de domaines monovalents (une valeur par domaine) ou les CSP n-aires cohérents avec exactement un tuple autorisé par relation. Néanmoins, les CSP qui ont une micro-structure (généralisée) qui est un graphe CSG^1 peuvent être cohérents ou non et il est facile de construire un CSP avec plusieurs solutions, ce qui correspond à une collection de cliques de taille n (cas binaire) ou e (cas n-aire). En outre, contrairement aux classes des sections précédentes, dans les graphes CSG^k (avec $k \geq 1$), il n'y a pas de restriction sur les valeurs de n, d, e, a ou r. Toutefois, cet ensemble de classes de CSP doit encore être étudié en détail pour évaluer son intérêt pratique.

6 Discussion et Perspectives

Cet article portait sur l'analyse de la complexité temporelle des algorithmes génériques classiques de résolution de CSP dans une perspective nouvelle et surtout différente des études menées jusque'à présent. Notre analyse exprime la complexité en termes de nombre de cliques maximales dans la micro-structure (généralisée) du CSP à résoudre. Cette analyse révèle que pour l'essentiel, le backtracking et le forward checking visitent chaque clique maximale de la micro-structure (généralisée) au plus une fois.

A partir de cette analyse, nous déduisons des classes polynomiales de CSP qui peuvent être résolues par des algorithmes classiques en temps polynomial, *sans avoir à reconnaître que l'instance figure dans la classe.* Aussi, les résultats obtenus apportent un éclairage nouveau sur l'analyse de la complexité des CSP.

La première perspective de ce travail est constituée par l'étude de classes de graphes possédant un nombre polynomial de cliques maximales. L'étude de [16] revêt ici un intérêt particulier car elle a permis de caractériser précisément ces classes de graphes sur la base de graphes d'intersection.

Une autre perspective importante consiste à mettre en évidence des liens qui pourraient exister entre notre analyse et les classes polynomiales obtenues par des approches différentes. Il devrait être assez clair notamment que nos classes sont orthogonales aux classes polynomiales basées sur la structure. Par exemple, il n'existe aucune raison pour qu'un CSP arborescent ne possède pas un nombre exponentiel de cliques. Plus généralement, les classes basées sur la structure nécessitent généralement des algorithmes dédiés, et les algorithmes de backtracking génériques sont de complexité polynomiales sur elles uniquement s'ils procèdent, par exemple, via l'exploitation d'un ordre d'affectation des variables spécifique. Par contre, pour les classes définies par un langage de contraintes, il serait possible que certaines soient capturées par notre analyse, tout comme c'est le cas pour le problème de satisfiabilité dans les travaux de [15]. Enfin, les classes hybrides sont beaucoup plus proches dans l'esprit de notre approche

et l'étude en profondeur des liens entre ces classes et notre analyse constitue donc une perspective naturelle à court terme.

En ce qui concerne les solveurs, la complexité d'algorithmes tels que MAC, dont le comportement est différent de ceux qui sont analysés ici, doit également être étudiée, car cette complexité ne s'inscrit pas naturellement dans l'évaluation que nous avons proposée.

Enfin, une perspective importante porte sur l'extension de notre étude aux autres généralisations possibles de la micro-structure pour le cas des problèmes de satisfaction de contraintes non-binaires.

7 Remerciements

Les auteurs tiennent à remercier les relecteurs anonymes de cet article qui ont permis déjà d'en améliorer la rédaction mais aussi, qui ont suggéré des pistes intéressantes pour la poursuite de ce travail.

Références

[1] C. Bessière, P. Meseguer, E. C. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. *Artificial Intelligence*, 141 :205–224, 2002.

[2] A. Chmeiss and P. Jégou. A generalization of chordal graphs and the maximum clique problem. *Information Processing Letters*, 62 :111–120, 1997.

[3] David A. Cohen. A New Classs of Binary CSPs for which Arc-Constistency Is a Decision Procedure. In *Proceedings of CP 2003*, pages 807–811, 2003.

[4] M. Cooper, D. Cohen, and P. Jeavons. Characterising Tractable Constraints. *Artificial Intelligence*, 65(2) :347–361, 1994.

[5] M. Cooper, Peter Jeavons, and Andras Salamon. Generalizing constraint satisfaction on trees : hybrid tractability and variable elimination. *Artificial Intelligence*, 174 :570–584, 2010.

[6] R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38 :353–366, 1989.

[7] Vida Dujmovic, Gasper Fijavz, Gwenaël Joret, Thom Sulanke, and David R. Wood. On the maximum number of cliques in a graph embedded in a surface. *European J. Combinatorics*, 32(8) :1244–1252, 2011.

[8] E. Freuder. A Sufficient Condition for Backtrack-Free Search. *JACM*, 29 (1) :24–32, 1982.

[9] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[10] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :343–282, 2000.

[11] R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14 :263–313, 1980.

[12] P. Jégou. Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In *Proceedings of AAAI 93*, pages 731–736, Washingtown, DC, 1993.

[13] J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3 :23–28, 1965.

[14] B. Nadel. *Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms*, pages 287–342. In *Search in Artificial Intelligence*. Springer-Verlag, 1988.

[15] Antoine Rauzy. Polynomial restrictions of SAT : What can be done with an efficient implementation of the Davis and Putnam's procedure. In U. Montanari and F. Rossi, editors, *Proc. International Conference on Principles of Constraint Programming (CP 1995)*, pages 515–532. Springer Verlag, 1995.

[16] Bill Rosgen and Lorna Stewart. Complexity results on graphs with few cliques. *Discrete Mathematics and Theoretical Computer Science*, 9 :127–136, 2007.

[17] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 550–556, 1990.

[18] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[19] D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of ECAI*, pages 125–129, 1994.

[20] David R. Wood. On the maximum number of cliques in a graph. *Graphs and Combinatorics*, 23 :337–352, June 2007.

Extraction de Motifs sous Contraintes Quantifiées

Mehdi Khiari[1] **Arnaud Lallouet**[1] **Jérémie Vautard**[2]

[1] GREYC (CNRS - UMR 6072) – Université de Caen Basse-Normandie
Boulevard du Maréchal Juin, 14000 Caen
[2] Chercheur indépendant
[1]{prenom.nom}@unicaen.fr [2]jeremie.vautard@gmail.com

Résumé

Au cours des dernières années, des approches d'extraction de motifs en fouille de données utilisant la PPC on été proposées. Ces approches ont montré leur utilité pour modéliser de manière flexible une large panoplie de contraintes, notamment les contraintes portant sur plusieurs motifs. Néanmoins, ces approches se basent sur les CSPs où toutes les variables sont quantifiées existentiellement. Or certaines requêtes n-aires (requêtes portant sur plusieurs motifs) requièrent la quantification universelle pour être modélisées de manière concise et élégante, comme par exemple la requêtes peak (un motif est considéré comme pic si tous ses voisins ont une valeur, par rapport à une mesure, inférieure à un seuil donné). Nous proposons dans cet article un cadre générique permettant la modélisation et la résolution de problèmes d'extraction de motifs sous contraintes quantifiées.

1 Introduction

L'Extraction de Connaissances dans les Bases de Données (ECBD) a pour objectif la découverte d'informations utiles et pertinentes répondant aux intérêts de l'utilisateur. L'extraction de motifs sous contraintes est un cadre proposant des approches et des méthodes génériques pour la découverte de motifs locaux [3]. Mais, ces méthodes ne prennent pas en considération le fait que l'intérêt d'un motif dépend souvent d'autres motifs et que les motifs les plus recherchés par l'utilisateur sont fréquemment noyés parmi une information volumineuse et redondante. C'est pourquoi la transformation des collections de motifs locaux en modèles globaux tels que les classifieurs ou le clustering [16, 10] est une voie active de recherche et la découverte de motifs sous contraintes portant sur des combinaisons de motifs locaux est un problème majeur. Dans la suite, ces contraintes sont appelées *contraintes n-aires* et les motifs concernés, *motifs n-aires*.

Il y a encore quelques années, peu de travaux concernant l'extraction de motifs n-aires ont été menés et les méthodes développées sont toutes ad hoc [23, 18]. Ceci est expliqué par la difficulté de la tâche due à l'ampleur de l'espace de recherche lorsqu'il s'agit d'extraire des motifs sous contraintes n-aires. Ce manque de généricité est un frein à la découverte de motifs pertinents et intéressants car chaque contrainte n-aire entraîne la conception et le développement d'une méthode ad hoc.

Au cours des quatre dernières années, des approches génériques d'extraction de motifs sous contraintes n-aires sont proposées [14, 11, 19]. Ces approches utilisent différentes méthodes de résolution. [14] utilise uniquement un solveur de CSP en proposant une modélisation sous forme de CSP du problème d'extraction de motifs sous contraintes n-aires. [11] propose un solveur dédié à l'extraction de motifs dont les méthodes de filtrage et de propagation sont inspirés d'algorithmes issus de la communauté PPC notamment AC-5 [12]. Enfin, [19] propose une méthode d'extraction utilisant un solveur SAT. Ces différentes approches sont associées à des langages de contraintes permettant à l'utilisateur de spécifier ses requêtes de manière déclarative. Ce dernier a ainsi plus de facilité à cibler des motifs de haut niveau plus faciles à interpréter.

Certaines requêtes n-aires (i.e., requêtes comportant des contraintes n-aires) requièrent en plus la quantification universelle pour être modélisées de manière concise et élégante, comme par exemple la requêtes *peak* [7] (un motif est considéré comme pic si tous ses voisins ont une valeur, par rapport à une mesure, inférieure à un seuil donné). On ne trouve dans la littérature que la description de la requête sans méthode de résolution associée. Ce type de requêtes n'est pas

Trans.	Items					
t_1	A	B	C		c_1	
t_2	A	B		D	c_1	
t_3	A	B		D	c_1	
t_4	A	B			c_1	
t_5		B		D		c_2
t_6	A	B	C	D		c_2
t_7	A					c_2

TABLE 1 – Jeu de données exemple

supporté par les méthodes génériques précédemment citées.

Nous proposons dans cet article un cadre générique permettant la modélisation et la résolution de problèmes d'extraction de motifs sous contraintes quantifiées.

L'article est organisé comme suit : la section 2 introduit le cadre général de l'extraction de motifs sous contraintes et la nécessité de développer des méthodes permettant l'extraction de motifs de haut niveau. Nous y présentons aussi deux requêtes n-aires d'extraction de motifs sous contraintes quantifiées. La section 3 trace un état de l'art des approches d'extraction de motifs utilisant la PPC et en particulier l'approche *Pure-CP*. La section 4 présente les cadre des QCSP et des QCSP$^+$. Les sections 5 et 6 proposent respectivement la nouvelle approche que nous proposons et l'étude expérimentale associée.

2 Extraction de motifs sous contraintes

2.1 Contexte et définitions

Soit $\mathcal{I}$ un ensemble de littéraux distincts appelés *items*, un motif ensembliste [1] d'items correspond à un sous-ensemble non vide de $\mathcal{I}$. Ces motifs sont regroupés dans le langage $\mathcal{L}_\mathcal{I} = 2^\mathcal{I} \backslash \emptyset$. Un contexte transactionnel est alors défini comme un multi-ensemble de motifs de $\mathcal{L}_\mathcal{I}$. Chacun de ces motifs, appelé transaction, constitue une entrée de la base de données.

Ainsi, le tableau 1 présente un contexte transactionnel $\boldsymbol{r}$ où 7 transactions étiquetées $t_1, \ldots, t_7$ sont décrites par 6 items $A, \ldots, D, c_1, c_2$.

L'extraction de motifs a pour but la découverte d'informations à partir de tous les motifs ou d'un sous ensemble de $\mathcal{L}_\mathcal{I}$. L'extraction sous contraintes cherche la collection de tous les motifs de $\mathcal{L}_\mathcal{I}$ présents dans $\boldsymbol{r}$ et satisfaisant un prédicat appelé *contrainte*. Ces motifs sont appelés *motifs locaux* ; ce sont des régularités observées dans certaines parties des données. La localité de ces motifs provient du fait que, vérifier s'ils satisfont une contrainte donnée, peut s'effectuer indépendamment des autres motifs.

Les notions de support et de fréquence d'un motif sont définies comme suit :

Définition 2.1 (Support, fréquence)
Une transaction $t \in \mathcal{T}$ supporte le motif X (ou X est présent dans t) ssi $X \subseteq t$ (ou $\forall i \in X, R(i,t) = 1$). Le ***support*** $\texttt{support}(X)$ *d'un motif X d'attributs est l'ensemble des transactions qui le supportent.*
Sa mesure de fréquence $\texttt{freq}(X)$ *est le cardinal du support.*

Il y a de nombreuses contraintes permettant d'évaluer la pertinence et la qualité des motifs locaux. Un exemple bien connu est celui de la contrainte de fréquence qui permet de rechercher les motifs X ayant une fréquence $\texttt{freq}(X)$ supérieure à un seuil minimal fixé $min_{\texttt{freq}} > 0$. De nombreux travaux [20] remplacent la fréquence par d'autres mesures permettant d'évaluer l'intérêt des motifs locaux recherchés. C'est le cas de la mesure d'aire : soit X un motif, $\texttt{area}(X)$ est le produit de la fréquence de X par sa taille, i.e., $\texttt{area}(X) = \texttt{freq}(X) \times \texttt{size}(X)$ où $\texttt{size}(X)$ désigne la longueur (i.e., le nombre d'items) de X.

En pratique, l'utilisateur est souvent intéressé par la découverte de motifs de plus haut niveau, comme par exemple les règles de classification les plus simples afin d'éviter le sur-apprentissage [26], ou encore des paires de règles d'exception [23] capables de révéler des caractéristiques globales des données. De tels motifs reposent sur des propriétés impliquant plusieurs motifs locaux et sont formalisés par la notion de *contrainte n-aire*.

Définition 2.2 (Contrainte n-aire) *Une contrainte n-aire est une contrainte portant sur n motifs.*

Définition 2.3 (Requête n-aire) *Une requête n-aire est une conjonction de contraintes contenant au moins une contrainte n-aire.*

2.2 Extraction de motifs de haut niveau

Un problème majeur dans les processus de l'ECBD est le nombre conséquent de motifs produits rendant leur utilisation difficile. Ainsi, les motifs les plus significatifs sont souvent noyés au milieu d'informations triviales ou redondantes. Ce problème est souvent rencontré par les extracteurs de motifs locaux dont le manque d'expressivité est frein à la découverte de motifs plus pertinents.

Ainsi plusieurs travaux se sont intéressés à réduire le nombre de motifs extraits pour obtenir des motifs plus pertinents. Plusieurs pistes se sont dégagées.

1. Dans cet article, nous nous intéressons au cas des motifs ensemblistes

[17, 5, 6, 25] proposent des méthodes permettant de réduire la redondance dans l'ensemble de motifs extraits, mais, même si ces approches comparent explicitement les motifs locaux entre eux, elles sont principalement fondées sur la réduction de la redondance entre motifs ou poursuivent des objectifs spécifiques tels que la classification.

[18, 22] présentent des approchent d'extraction de motifs dédiées à des classes particulières de contraintes n-aires. D'autres travaux se sont intéressés à des requêtes n-aires particulières pour lesquelles il proposent des méthodes de résolution ad hoc.

Enfin, très récemment, des approches génériques d'extraction de motifs sous contraintes n-aires sont proposées : l'idée clé de ses approches est de se baser sur un langage de contraintes à partir duquel il est possible de modéliser une très large panoplie de requêtes n-aires [14, 11, 19]. Néanmoins, aucune de ces approches ne permet de modéliser des requêtes contenant des contraintes quantifiées.

2.3 Motifs pics (peak)

Les motifs *pic* ou sommet (ou encore *peak*) [7] sont un exemple de requête n-aire se modélisant à l'aide de contraintes quantifiées.

Un motif pic est un motif qui affiche une valeur, par rapport à une mesure m donnée, assez grande par rapport à celles affichées par tous ses voisins. En d'autres termes, un motif pic a un comportement exceptionnel par rapport à ses voisins (le voisinage d'un motif est calculé par rapport à une distance d donnée).

Soit $m : \mathcal{L}_\mathcal{I} \rightarrow \mathbb{R}$ une mesure, δ un entier et ρ un réel, et soit d une distance, on a :

$$peak(X) \equiv \begin{cases} vrai & \text{si } \forall Y \in \mathcal{L}_\mathcal{I} \ tq \ d(X,Y) < \delta, \\ & \quad m(X) \geq \rho \times m(Y) \\ faux & \text{sinon} \end{cases}$$

Cette requête a été présentée dans [7] sans méthode de résolution associée.

Les motifs pics peuvent être recherchés relativement à diverses mesures comme l'aire, le taux de croissance des motifs émergents, etc. Les motifs émergents [9] sont des motifs dont la fréquence varie fortement entre deux classes. Ils caractérisent les classes de manière quantitative et qualitative. De par leur capacité à faire ressortir les distinctions entre classes, les motifs émergents permettent de construire des classifieurs ou de proposer une aide au diagnostic. L'émergence d'un motif est quantifiée par la mesure de son taux de croissance (*growth rate*) entre deux classes.

Définition 2.4 (Taux de croissance d'un motif)

Soient D_1et $D_2 \subseteq \mathcal{T}$ les ensembles de transactions correspondantes respectivement aux classes c_1 et c_2 et soit $\texttt{freq'}(X, D_j)$ la fréquence du motif X dans D_j, le taux de croissance de X dans D_1 est :

$$\texttt{growth-rate}_{D_1}(X) = \frac{|D_2| \times \texttt{freq'}(X,D_1)}{|D_1| \times \texttt{freq'}(X,D_2)}$$

Définition 2.5 (Motif émergent)

Soient un taux de croissance minimum ρ et une base de données $\mathbf{r}$ partitionnée en deux sous ensembles D_1 et D_2. Le motif X est émergent de D_2 vers D_1 ssi $\texttt{growth-rate}_{D_1}(X) \geq \rho$

Exemple 2.1 (pic d'émergence)

Soit $m(X) = \texttt{growth-rate}_{D_1}(X)$, $\delta = 1$, $\rho = 2$ et $d(X,Y) = |X \backslash Y| + |Y \backslash X|$, et considérons le jeu de données décrit par la table 1.

Le motif **AB** *est un peak puisque $\texttt{growth-rate}_{D_1}(AB) = 3$ est au moins 2 fois plus grand que les taux de croissance de ses 4 voisins A, B, ABC et ABD qui sont respectivement* 1.5, 1.5, 0.75 *et* 1.5.

2.4 Groupes de synexpression

Le domaine de l'analyse d'expression de gènes fournit un autre exemple de contrainte n-aire.

En effet, les motifs locaux composés de tags (ou gènes) et satisfaisants la contrainte d'aire sont au cœur de la recherche de groupes de synexpression [15]. Néanmoins, dans des données réelles telles que celles du transcriptome, la recherche de motifs tolérants aux erreurs [2] y est capitale afin de tenir compte de l'incertain qui est présent dans les données [2]. Les contraintes n-aires sont une façon naturelle de concevoir des motifs tolérants aux erreurs candidats à être des groupes de synexpression : ceux-ci sont définis par la combinaison de plusieurs motifs locaux satisfaisant la contrainte d'aire et ayant un fort recouvrement entre eux. Plus précisément, à partir de deux motifs locaux X et Y, on définit la contrainte n-aire suivante $\texttt{area}(X) > min_{\texttt{area}} \wedge \texttt{area}(Y) > min_{\texttt{area}} \wedge (\texttt{area}(X \cap Y) > \alpha \times min_{\texttt{area}})$ où $min_{\texttt{area}}$ est le seuil d'aire minimale et α est un paramètre donné par l'utilisateur pour fixer le recouvrement minimal entre motifs locaux. Cette contrainte n-aire peut être étendue à k motifs ($k \geq 2$) comme suit :

2. Un concept formel associe un ensemble maximal d'items à un ensemble maximal de transactions, des motifs tolérants aux erreurs constituent une relaxation de cette association en cherchant à extraire des rectangles d'ensembles denses en valeur 1 mais acceptant aussi un nombre contrôlé de 0.

$Synexpression(X_1, ..., X_k) \equiv$

$$\begin{cases} \forall\ 1 \leq i < j \leq k, \\ \texttt{area}(X_i) > min_{\texttt{area}} \wedge \\ \texttt{area}(X_j) > min_{\texttt{area}} \wedge \\ \texttt{area}(X_i \cap X_j) > \alpha \times min_{\texttt{area}} \wedge \\ \forall Z\ tq\ [\texttt{freq}(Z) > min_{\texttt{freq}}\ et\ \texttt{area}(Z) > min_{\texttt{area}}], \\ \exists i \in [1..k], \texttt{area}(Z \cap X_i) < \alpha \times min_{\texttt{area}} \end{cases}$$

L'utilisation de la quantification universelle dans la dernière contrainte permet de vérifier que le regroupement trouvé est maximal.

3 La PPC pour l'extraction de motifs

3.1 État de l'art

En 2008, [8] a proposé une approche permettant de traiter, dans un cadre unifié, un large ensemble de motifs locaux et de contraintes telles que la fermeture, la maximalité, les contraintes monotones ou anti-monotones et des variations de ces contraintes. Cette approche, qui marque un cadre fondateur, est malgré tout limitée aux motifs locaux.

Dans [14, 13], nous avons proposé une approche générique pour l'extraction de motifs sous contraintes n-aires. Cette approche (appelée *Pure-CP*) repose uniquement sur l'utilisation d'un solveur de CSP. Elle étend la modélisation proposée dans [8].

[21] propose une comparaison entre les performances des solveurs des CSP génériques et les extracteurs de motifs classiques. Il présente aussi un nouvel algorithme d'extraction de motifs sous contraintes basé sur les techniques de la PPC. [11] présente une méthode pour modéliser et résoudre des problèmes d'extraction portant sur k motifs ce qui est similaire aux contraintes n-aires.

Dans [19], nous avons proposé un langage de contraintes permettant d'écrire des requêtes n-aires de manière déclarative. L'ensemble des contraintes primitives composant ce langage peut être modélisé et résolu utilisant *Pure-CP*.

3.2 Approche Pure-CP

Nous avons proposé dans [14] *Pure-CP*, une approche générique pour l'extraction de motifs sous contraintes n-aires utilisant uniquement un solveur de CSP. Cette section décrit les principales étapes de cette approche.

a) Modélisation

Soit $\boldsymbol{r}$ un contexte transactionnel où $\mathcal{I}$ est l'ensemble de ses n items et $\mathcal{T}$ l'ensemble de ses m transactions. Soit $\boldsymbol{d}$ la matrice 0/1 de dimension (m, n) telle que $\forall t \in \mathcal{T}, \forall i \in \mathcal{I}$, $(d_{t,i} = 1)$ ssi $(i \in t)$.

Soient $X_1, X_2, ..., X_k$ les k motifs recherchés. Chaque motif inconnu X_j est modélisé par n variables de décision $\{X_{1,j}, X_{2,j}, ..., X_{n,j}\}$ (de domaine $\{0, 1\}$) telles que $(X_{i,j} = 1)$ ssi l'item i appartient au motif X_j. Le support T_{X_j} de chaque motif X_j est représenté par m variables de décision $\{T_{1,j}, T_{2,j}, ..., T_{m,j}\}$ (de domaine $\{0, 1\}$) qui sont associées à X_j comme suit : $(T_{t,j} = 1)$ ssi $(X_j \subseteq t)$.

b) Liens entre les variables et la base de données

La relation entre chaque motif recherché X_j ($1 \leq j \leq k$), son support T_{X_j} et la base de données $\boldsymbol{r}$est établie via des contraintes réifiées imposant que, pour chaque transaction t, $(T_{t,j} = 1)$ ssi $(X_j \subseteq t)$.

Ceci est formulé par l'équation suivante :

$$\forall j \in [1..k], \forall t \in \mathcal{T}, (T_{t,j} = 1) \Leftrightarrow \sum_{i \in \mathcal{I}} X_{i,j} \times (1 - d_{t,i}) = 0 \quad (1)$$

c) Reformulation des contraintes

Considérons un opérateur $\texttt{op} \in \{<, \leq, >, \geq, =, \neq\}$; les contraintes numériques se reformulent comme suit :

- $\texttt{freq}(X_p)\ \texttt{op}\ \alpha \rightarrow \sum_{t \in \mathcal{T}} T_{t,p}\ \texttt{op}\ \alpha$
- $\texttt{size}(X_p)\ \texttt{op}\ \alpha \rightarrow \sum_{i \in \mathcal{I}} X_{i,p}\ \texttt{op}\ \alpha$

Certaines contraintes ensemblistes (telles que égalité, inclusion, appartenance,...) se reformulent directement à l'aide de contraintes linéaires :

- $X_p = X_q \rightarrow \forall i \in \mathcal{I}, X_{i,p} = X_{i,q}$
- $X_p \subseteq X_q \rightarrow \forall i \in \mathcal{I}, X_{i,p} \leq X_{i,q}$
- $i_o \in X_p \rightarrow X_{i_0,p} = 1$

Les autres contraintes ensemblistes (telles que intersection, union, différence,...) se reformulent aisément à l'aide de contraintes booléennes en utilisant la fonction de conversion (b::$\{0, 1\} \rightarrow \{False, True\}$) et les opérateurs booléens usuels :

- $X_p \cap X_q = X_r \rightarrow \forall i \in \mathcal{I}, b(X_{i,r}) = b(X_{i,p}) \wedge b(X_{i,q})$
- $X_p \cup X_q = X_r \rightarrow \forall i \in \mathcal{I}, b(X_{i,r}) = b(X_{i,p}) \vee b(X_{i,q})$
- $X_p \backslash X_q = X_r \rightarrow \forall i \in \mathcal{I}, b(X_{i,r}) = b(X_{i,p}) \wedge \neg b(X_{i,q})$

3.3 Discussion

Les variables d'un CSP sont toutes quantifiées existentiellement. Dans le cadre d'extraction de motifs sous contraintes utilisant une modélisation CSP, les seuls cas envisageables sont :

- **Existe-il** un motif vérifiant une propriété locale P (motifs locaux) ?
- **Existe-il** un motif vérifiant une propriété P' définie par rapport à un ensemble de motifs préalablement connus ou faisant partie des inconnues du problème (motifs n-aires) ?

Mais, dans les problèmes d'extraction de motifs sous contraintes, on trouve souvent des contraintes du type :

– **Existe-il** un motif X vérifiant une propriété P **quel que soit** un motif Y vérifiant une propriété P' (P' est généralement définie en fonction de X) ?

Pour la modélisation de telles requêtes, nous avons besoin d'utiliser la *quantification universelle* ($\forall$) que les CSPs classiques ne peuvent pas gérer. D'où le recours à l'utilisation des Problèmes de Satisfaction de Contraintes Quantifiées (QCSP) qui sont certes plus difficiles à résoudre mais strictement plus expressifs que les CSPs.

4 CSPs Quantifiés

4.1 QCSP

La formule logique associée à un CSP ayant pour variables $x_1, x_2, ..., x_n$ est de la forme :

$$\exists x_1 \in D_1, \exists x_2 \in D_2, ..., \exists x_n \in D_n \; c_1 \wedge c_2 \wedge ... \wedge c_k$$

Cette formule est en *forme normale prénexe*. Toutes les variables apparaissent ainsi quantifiées en début de formule, constituant le *préfixe*, chaque quantification liant toutes les occurrences de la variable dans le reste de la formule. Les QCSP étendent les CSP en introduisant dans cette formule des quantificateurs universels (i.e., $\forall$) en lieu et place de certains des quantificateurs existentiels (i.e., $\exists$) présents[4]. Par exemple, on peut représenter la formule suivante :

$$\forall x_1 \in D_1, \exists x_2 \in D_2, ..., \exists x_n \in D_n \; c_1 \wedge c_2 \wedge ... \wedge c_k$$

Cette formule reste en forme normale prénexe. Cependant, l'introduction de quantificateurs universels implique par ailleurs une relation d'ordre sur les variables qui n'était pas présente dans les CSP : la satisfiabilité de la formule logique précédente est susceptible de changer si l'on inverse dans le préfixe deux variables quantifiées de manière différente ou même deux variables séparées par une alternance de quantificateurs. Cette relation d'ordre sera donc aussi précisée dans la définition formelle d'un QCSP. Cependant, si deux variables qui se suivent dans le préfixe sont quantifiées de la même manière, leur ordre relatif n'influe aucunement sur la sémantique de la formule. Une sous séquence du préfixe où toutes les variables sont quantifiées de la même manière est appelée *bloc de quantification* ou *qset*.

Dans les définitions qui suivent, soit p un entier tel que $p \leq n$ (où n est le nombre de variables du problème).

Définition 4.1 (qset)
Un qset est un couple (qt, W) avec W un ensemble de variables et $qt \in \{\forall, \exists\}$ un quantificateur.

Le préfixe d'un QCSP peut donc être formellement défini comme suit :

Définition 4.2 (préfixe de QCSP)
Un préfixe de QCSP est une séquence de qsets $((qt_1, W_1), ..., (qt_p, W_p))$ dont les ensembles de variables sont deux à deux disjoints ($\forall i, j,\ i \neq j \rightarrow (W_i \cap W_j = \emptyset)$).

Le QCSP se définit alors comme un préfixe accompagné d'un ensemble de contraintes dont les variables doivent faire partie du préfixe :

Définition 4.3 (QCSP)
Un QCSP est un couple (P, G) avec $P = ((qt_1, W_1), ..., (qt_p, W_p))$ un préfixe et G un ensemble de contraintes appelé Goal tel que $var(G) \subseteq \bigcup_{i \in \{1,...,p\}} W_i$.

Exemple 4.1
Le problème :

$\exists X \in \{0,1,2,3\}$, $\forall Y \in \{0,1,2\}$, $\exists Z \in \{0,\ldots,6\}$
$X + Y = Z$

est représenté par le QCSP suivant :
$Q = ([(\exists, X), (\forall, Y), (\exists, Z,)], \{X + Y = Z\})$

4.2 QCSP$^+$

Le formalisme des QCSP est étendu dans [1] pour permettre de modéliser explicitement des restrictions propres à chaque variable quantifiée. Chaque restriction est définie sous forme d'un ensemble de contraintes : le quantificateur ne porte alors que sur les tuples satisfaisant ces contraintes.

Un QCSP$^+$ doit donc intégrer des contraintes dont le but est de réduire la portée des quantificateurs. Ces contraintes sont donc ajoutées directement dans les qset.

Définition 4.4 (rqset)
Un rqset est un triplet (qt, W, C) avec W un ensemble de variables, $qt \in \{\forall, \exists\}$ un quantificateur et C un ensemble de contraintes.

Le préfixe d'un QCSP$^+$ peut donc être formellement défini comme suit :

Définition 4.5 (préfixe de QCSP$^+$)
Un préfixe de QCSP est une séquence de rqsets $((qt_1, W_1, C_1), ..., (qt_p, W_p, C_p))$ telle que :
– $\forall i, j \in \{1, .., p\}, i \neq j \rightarrow W_i \cap W_j = \emptyset$
– $\forall i \in \{1, .., p\}, var(C_i) \subseteq (W_1 \cup ... \cup W_p)$

Le QCSP$^+$ se définit alors de manière analogue au QCSP comme suit :

Définition 4.6 (QCSP$^+$)
Un QCSP$^+$ est un couple (P, G) avec $P = ((qt_1, W_1, C_1), ..., (qt_p, W_p, C_p))$ un préfixe de QCSP$^+$ et G un ensemble de contraintes appelé Goal tel que $var(G) \subseteq \bigcup_{i \in \{1,...,p\}} W_i$.

Exemple 4.2
Le problème :

$\exists X \in \{0, 1, 2, 3\}$
$\forall Y \in \{0, 1, 2\}$ *tel que* $Y \neq X$,
$\exists Z \in \{0, \ldots, 6\}$ *tel que* $Z < 2 * X$,
$X + Y = Z$

est représenté par le QCSP$^+$ suivant :
$Q = \big([(\exists, X), (\forall, Y, \{X \neq Y\}),$
$(\exists, Z, \{Z < 2 * X\})], Goal = \{X + Y = Z\}\big)$

4.3 *QeCode*

QeCode[3] est un solveur de QCSP écrit en C++ et basé sur le solveur de contraintes *Gecode*[4]. Il intègre aussi les procédures de recherche des QCSP$^+$ proposées dans [24]. L'ensemble des contraintes implémentées dans *Gecode* (y compris par les utilisateurs, *Gecode* permettant l'utilisation de propagateurs personnalisés) peut donc être utilisé dans *QeCode*. Ceci est à l'origine du choix de *QeCode* pour la mise en œuvre de l'approche que nous proposons dans cet article.

5 QCSP$^+$ pour l'extraction de motifs

L'approche d'extraction de motifs sous contraintes quantifiées que nous proposons (approche QCSP) repose sur la modélisation proposée pour *Pure-CP*. Elle utilise les mêmes contraintes réifiées pour établir le lien entre les motifs recherchés et les transactions les supportant (cf. équation 1). Elle partage aussi la modélisation des motifs recherchés ainsi que l'implantation et la reformulation des contraintes (cf. section 3.2).

Ainsi, soit $\boldsymbol{r}$ un contexte transactionnel où $\mathcal{I}$ est l'ensemble de ses n items et $\mathcal{T}$ l'ensemble de ses m transactions et soient $X_1, X_2, ..., X_k$ les k motifs recherchés. Chaque motif inconnu X_j est modélisé par n variables de décision $\{X_{1,j}, X_{2,j}, ..., X_{n,j}\}$ et son support T_{X_j} est représenté par m variables de décision $\{T_{1,j}, T_{2,j}, ..., T_{m,j}\}$.

Si un motif X_J est quantifié existentiellement, alors toutes ses variables de décision le seront aussi (comme dans *Pure-CP*). Par contre, si un motif X_j est quantifié universellement, alors toutes ses variables de décision seront quantifiées universellement.

3. `http://www.univ-orleans.fr/lifo/software/qecode/QeCode.html`
4. `http://http://www.gecode.org/`

La modélisation s'effectue par niveaux dépendant de la portée de chaque quantificateur, le dernier niveau étant consacré aux contraintes du *Goal*.

Remarque 5.1 *Toutes les variables que nous introduisons dans les modélisations qui suivent et pour lesquelles les domaines ne sont pas définis explicitement sont des variables représentant les motifs de type X_j et représentent donc en variables de décision de domaine $\{0, 1\}$ (cf. section 3.2).*

Exemple 5.1 (peak)
Cet exemple détaille la modélisation de la requête peak (cf. section 2.3).

Nous notons par $\boldsymbol{m}$ une mesure d'intérêt (taux d'émergence, aire, ...). Nous fixons à 1 la valeur de la distance maximale δ délimitant le voisinage.

Pour modéliser ce problème, on introduit les variables suivantes :
- *X_1 représentant le motif pic X, cette variable est quantifiée existentiellement.*
- *X_2 représentant le motif Y représentant le voisinage de X, cette variable est quantifiée universellement.*

$Q = \Big(\big[(\exists, X_1, C_1),\ (\forall, X_2, C_2)\big],\ Goal = C_3\Big)$

Où :
- $C_1 = \emptyset$
- $C_2 = d(X_1, X_2) \leq 1$
- $C_3 = \boldsymbol{m}(X_1) \geq \rho \times \boldsymbol{m}(X_2)$

Exemple 5.2 (Groupes de synexpression)
Nous présentons dans cet exemple la modélisation sous forme de QCSP$^+$ de la requête permettant la découverte de groupes de synexpression présentée dans la section 2.4. La quantification universelle apporte l'attrait supplémentaire de pouvoir extraire les groupements maximaux, renforçant ainsi l'hypothèse que l'ensemble des motifs agglomérés forme un groupe de synexpression.

Pour la modélisation de cette requête, nous introduisons les variables suivantes :
- *$\{X_1, X_2, ..., X_k, Z\}$ variables représentant les motifs,*
- *$\{A_2, A_3, ..., A_k, B\}$ variables représentant les intersections entre les motifs dans la perspective du calcul de l'aire de ces intersections,*
- *$\{i_3, i_4, ..., i_k, i\}$ variables compteurs. Ces variables permettent d'éviter d'associer une variable supplémentaire à chaque intersection deux à deux possibles entre les k motifs.*

La requête des groupes de synexpression permettant l'extraction de groupements de taille k $Synexpression(X_1, ..., X_k)$ peut ainsi être modélisée par la formule suivante :

$Synexpression(X_1, ..., X_k) \equiv$

$$
\begin{cases}
\exists X_1,\ [\mathit{freq}(X_1) > min_{freq}, \mathit{area}(X_1) > min_{area}] \\
\exists X_2, A_2\ tq\ \mathit{freq}(X_2) > min_{freq} \wedge \\
\qquad \mathit{area}(X_2) > min_{area} \wedge \\
\qquad X_1 \neq X_2\ \wedge\ A_2 = X_1 \cap X_2 \wedge \\
\qquad \mathit{area}(A_2) > \alpha \times min_{area} \\
\quad \exists X_3\ tq\ \mathit{freq}(X_3) > min_{freq} \wedge \\
\qquad \mathit{area}(X_3) > min_{area} \wedge X_1 \neq X_3 \wedge \\
\qquad X_2 \neq X_3 \\
\quad \forall i_3 \in [1..2], A_3\ tq\ A_3 = X_3 \cap X_{i_3} \wedge \\
\qquad \mathit{area}(X_3) > \alpha \times min_{area} \\
\quad ... \\
\quad \exists X_k\ tq\ \mathit{freq}(X_k) > min_{freq} \wedge \\
\qquad \mathit{area}(X_k) > min_{area} \wedge \\
\qquad X_1 \neq X_k\ \wedge\ ...\ \wedge\ X_{k-1} \neq X_k \\
\quad \forall i_k \in [1..k-1], A_k\ tq\ A_k = X_k \cap X_{i_k} \wedge \\
\qquad \mathit{area}(A_k) > \alpha \times min_{area} \\
\quad \forall Z\ tq\ \mathit{freq}(Z) > min_{freq} \wedge \\
\qquad \mathit{area}(Z) > min_{area} \wedge \\
\qquad X_1 \neq Z\ \wedge\ ...\ \wedge\ X_k \neq Z \\
\quad \exists i \in [1..k], B\ \ tq\ B = Z \cap X_i, \\
\quad \mathit{area}(B) < \alpha \times min_{area}
\end{cases}
$$

Nous considérons par exemple le cas où on veut spécifier un recouvrement composé de trois motifs X_1, X_2, X_3. La modélisation de la requête des synexpressions dans ce cas demande l'utilisation des variables suivantes :

- $\{X_1, X_2, X_3, Z\}$ *variables représentant les motifs.*
- $\{A_2, A_3, B\}$ *variables représentant les intersections entre les motifs,*
- $\{i_3, i\}$ *variables compteurs.*

La requête est finalement modélisée par le QCSP$^+$ suivant :

$Q = \Big(\big[(\exists, X_1, C_1), (\exists, (X_2, A_2), C_2), (\exists, (X_3), C_3),$
$(\forall, (i_3 \in [1..2], A_3), C_4), (\forall, (Z), C_5),$
$(\exists, (i \in [1..3], B), C_6) \big], Goal = C_7 \Big)$

où :

- $C_1 = \mathit{freq}(X_1) > min_{freq} \wedge \mathit{area}(X_1) > min_{area}$
- $C_2 = \mathit{freq}(X_2) > min_{freq} \wedge \mathit{area}(X_2) > min_{area} \wedge X_1 \neq X_2\ \wedge A_2 = X_1 \cap X_2\ \wedge \mathit{area}(A_2) > \alpha \times min_{area}$
- $C_3 = \mathit{freq}(X_3) > min_{freq} \wedge \mathit{area}(X_3) > min_{area} \wedge X_1 \neq X_2\ \wedge X_2 \neq X_3$
- $C_4 = A_3 = X_3 \cap X_{i_3}\ \wedge \mathit{area}(X_3) > \alpha \times min_{area}$
- $C_5 = \mathit{freq}(Z) > min_{freq} \wedge \mathit{area}(Z) > min_{area} \wedge X_1 \neq Z\ \wedge X_2 \neq Z\ \wedge X_3 \neq Z$
- $C_6 = B = Z \cap X_i$
- $C_7 = \mathit{area}(B) < \alpha \times min_{area}$

6 Expérimentations

Nous montrons, dans cette section, la faisabilité et les apports pratiques de l'approche basée sur les QCSPs présentée dans cet article.

6.1 Protocole expérimental

Nous avons mené différentes expérimentations sur plusieurs jeux de données de l'UCI *repository*[5] ainsi que sur le jeu de données réelles `Meningitis`[6]. La table 2 résume les différentes caractéristiques des jeux de données utilisés.

Les expérimentations ont été menées principalement sur la requête *peak* appliquée à la mesure du taux de croissance d'un motif émergent (`growth-rate`).

La machine utilisée est un PC 2.83 GHz Intel Core 2 Duo processor (4 GB de RAM) sous Ubuntu Linux.

Enfin nous avons utilisé le solveur de QCSP *QeCode*.

Jeu de données	#trans	#items	densité
`Australian`	690	55	0.25
`German`	1000	76	0.28
`Meningitis`	329	84	0.27
`Posto`	90	23	0.39

TABLE 2 – Description des jeux de données

6.2 Solveur utilisé

Nous utilisons le solveurs de QCSP *QeCode* (cf. section 4.3). Initialement, pour une instance QCSP donnée, *QeCode* retourne une seule stratégie gagnante (ou solution), si elle existe. Nous avons donc introduit quelques modifications sur le solveur afin d'obtenir une version retournant l'intégralité des solutions.

Ainsi, notre approche permet d'extraire l'ensemble correct et complet des motifs satisfaisant les requêtes contenant des motifs quantifiés universellement.

6.3 Résultats expérimentaux

Les figures 1 et 2 décrivent la variation du nombre de motifs pics détectés par rapport à la mesure du taux croissance `growth-rate` en fonction des paramètres ρ (cf. exemple 5.1) et $minfr$ (seuil minimal de fréquence des motifs émergents) pour les jeux de données `German`, `Meningitis`, `Postoperative-patient-data` (`Posto`) et `Australian`.

On constate que cette requête conduit à très peu de motifs (alors qu'il est bien commun que le nombre de motifs émergents soit très grand [9]). Ces résultats mettent en valeur l'efficacité de l'utilisation des contraintes contenant des quantifications universelles

5. `http://www.ics.uci.edu/~mlearn/MLRepository.html`
6. `Meningitis` recense les pathologies des enfants atteints d'une méningite virale ou bactérienne.

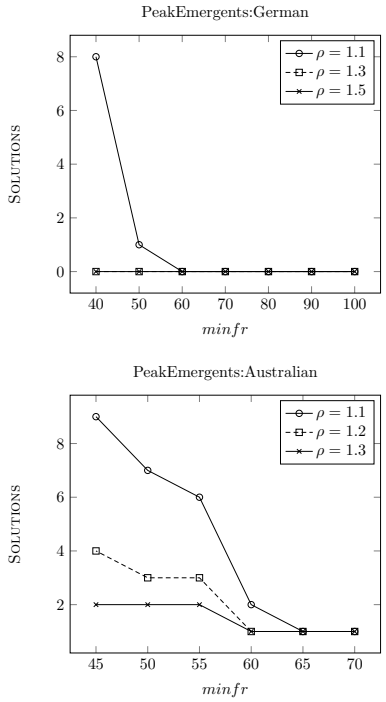


FIGURE 1 – Évolution du nombre de pics d'émergence extraits en fonction de $minfr$ (1/2)

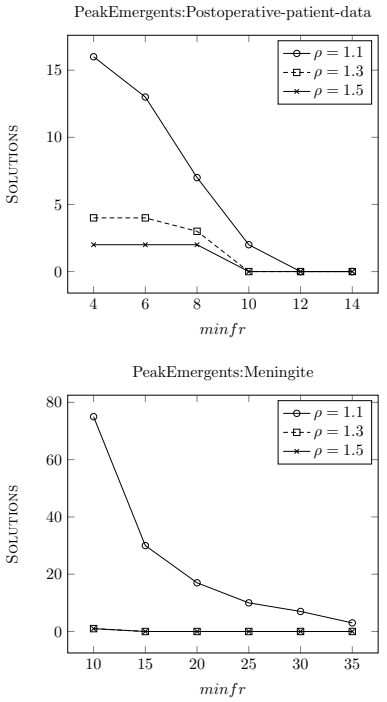


FIGURE 2 – Évolution du nombre de pics d'émergence extraits en fonction de $minfr$ (2/2)

dans la réduction du nombre de motifs extraits en définissant des propriétés fortes sur les motifs recherchés. Ceci ouvre une piste intéressante vers la définition de requêtes permettant de cibler des motifs de haut niveau et en faible nombre.

6.4 Efficacité

Les expérimentations menées permettent aussi de quantifier les temps de calcul consommés par notre approche (cf. figures 3 et 4). Naturellement, les temps de calcul varient en fonction de la taille des jeux de données et de la dureté des contraintes.

Les résultats expérimentaux montrent aussi que la difficulté à fouiller les jeux de données augmente en fonction du nombre d'items composant le jeu de données. Ceci est expliqué par le fait que, pour les requêtes utilisées dans les expérimentations, la quantification universelle porte sur les variables représentant les motifs. La figure 5 met l'accent sur ce comportement que nous expliquons plus en détail dans la section suivante.

7 Discussion

Dans le cas des QCSP, une simple affectation de toutes les variables ne permet pas de vérifier la viabilité de l'instanciation comme c'est le cas pour les CSP classiques : en effet, le *Goal* doit être vérifié pour toutes les valeurs possibles des variables universellement quantifiées du problème. Une solution d'un QCSP doit donc exprimer tous ces cas possibles. De plus, la formule logique associée (qui est une formule du premier ordre sous forme prénexe) lie les variables dans un ordre donné par le préfixe. Ainsi, la valeur d'une variable existentielle postérieure (selon l'ordre du préfixe) à une variable universelle sera fonction de la valeur de celle-ci, alors que dans l'ordre inverse, il est nécessaire d'avoir une valeur d'une variable existentielle consistante avec toutes les valeurs de la variable universelle placée postérieurement.

Ceci a été traduit dans nos expérimentations par le fait qu'on a beaucoup plus de difficultés à gérer les jeux de données quand le nombre d'items augmente. Dans les requêtes présentées dans cet article et celles utilisées dans les expérimentations, la quantification universelle porte sur les variables représentant les motifs. Il est ainsi plus coûteux de vérifier tous les motifs possibles quand le nombre d'items augmente. C'est ce qui explique l'absence d'expérimentations présentées sur les groupes de synexpression où l'extraction s'effectue généralement dans des bases de données comportant plusieurs milliers d'items.

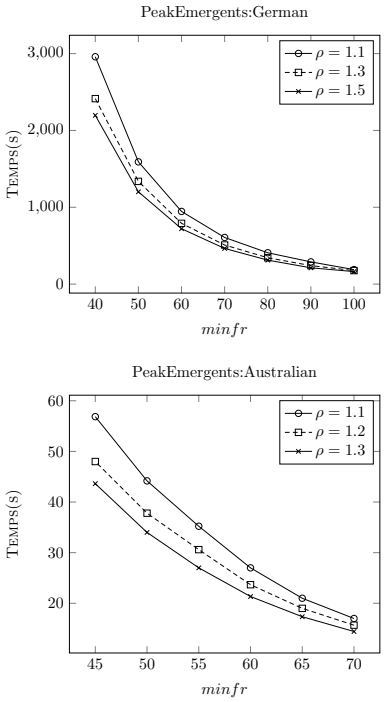


FIGURE 3 – Évolution du temps d'exécution de la requête d'extraction de pics d'émergence en fonction de $minfr$ (1/2)

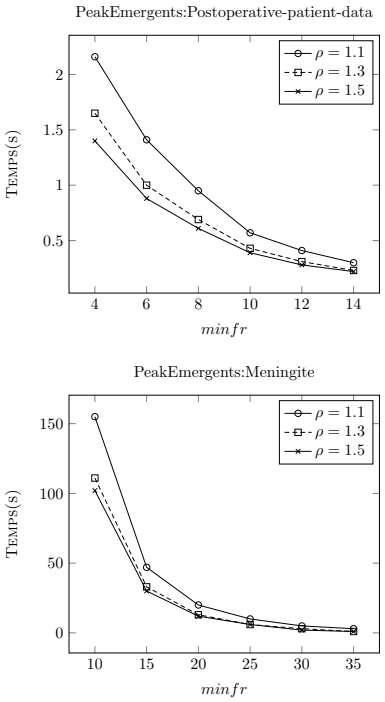


FIGURE 4 – Évolution du temps d'exécution de la requête d'extraction de pics d'émergence en fonction de $minfr$ (2/2)

8 Conclusion

L'approche proposée a montré l'élégance de la modélisation de certaines requêtes de fouilles de données utilisant les QCSPs. Acceptant l'intégralité de l'ensemble des contraintes primitives composant le langage de contraintes proposé dans [19], elle ouvre la porte vers la modélisation des requêtes cherchant à vérifier des propriété plus fortes sur les motifs et conduisant à l'extraction de motifs plus pertinent et facilement interprétables par l'utilisateur. Les résultats expérimentaux ont montré la faisabilité de l'approche même s'il n'est pas possible pour le moment de traiter des bases de données de tailles réelles (i.e, plusieurs milliers d'items) pour la recherche de groupes de synexpression. Nos perspectives s'inscrivent dans cette direction.

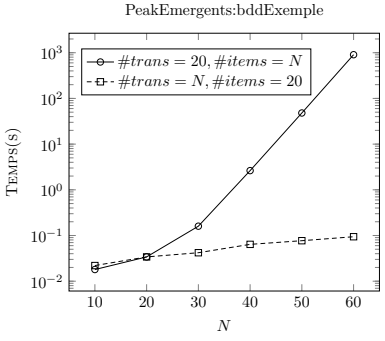


FIGURE 5 – Évolution du temps d'exécution de la requête d'extraction de pics d'émergence en fonction de la taille du jeu de données (densité fixée à 0.4)

Références

[1] Marco Benedetti, Arnaud Lallouet, and Jérémie Vautard. QCSP Made Practical by Virtue of Restricted Quantification. In *IJCAI*, pages 38–43. AAAI Press, 2007.

[2] J. Besson, C. Robardet, and J-F. Boulicaut. Mining a new fault-tolerant pattern type as an alternative to formal concept discovery. In *ICCS'06*, pages 144–157, Aalborg, Denmark, 2006.

[3] F. Bonchi, F. Giannotti, C. Lucchese, S. Orlando, R. Perego, and R. Trasarti. A constraint-based querying system for exploratory pattern discovery. *Inf. Syst.*, 34(1) :3–27, 2009.

[4] Lucas Bordeaux, Marco Cadoli, and Toni Mancini. CSP properties for quantified constraints : Definitions and complexity. In *National Conference on Artificial Intelligence*, pages 360–365. AAAI Press, 2005.

[5] Björn Bringmann and Albrecht Zimmermann. The chosen few : On identifying valuable patterns. In *ICDM*, pages 63–72. IEEE Computer Society, 2007.

[6] Björn Bringmann and Albrecht Zimmermann. One in a million : picking the right patterns. *Knowl. Inf. Syst.*, 18(1) :61–81, 2009.

[7] Bruno Crémilleux and Arnaud Soulet. Discovering knowledge from local patterns with global constraints. In *ICCSA (2)*, pages 1242–1257, 2008.

[8] Luc De Raedt, Tias Guns, and Siegfried Nijssen. Constraint Programming for Itemset Mining. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining edition :14 location*, Las Vegas, Nevada, USA, 2008.

[9] Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns : Discovering trends and differences. In *KDD*, pages 43–52, 1999.

[10] Arnaud Giacometti, Eynollah Khanjari Miyaneh, Patrick Marcel, and Arnaud Soulet. A framework for pattern-based global models. In *IDEAL*, volume 5788 of *Lecture Notes in Computer Science*, pages 433–440. Springer, 2009.

[11] Tias Guns, Siegfried Nijssen, and Luc De Raedt. k-Pattern set mining under constraints. *IEEE Transactions on Knowledge and Data Engineering*, 2011.

[12] Pascal Van Hentenryck, Yves Deville, and Choh-Man Teng. A generic arc-consistency algorithm and its specializations. *Artif. Intell.*, 57(2-3) :291–321, 1992.

[13] M. Khiari, P. Boizumault, and B. Crémilleux. Extraction de motifs n-aires utilisant la PPC. In *6èmes Journées Francophones de Programmation par Contraintes (JFPC'10)*, pages 167–176, Caen, 2010.

[14] Mehdi Khiari, Patrice Boizumault, and Bruno Crémilleux. Constraint programming for mining n-ary patterns. In David Cohen, editor, *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 552–567. Springer, 2010.

[15] J. Kléma, S. Blachon, A. Soulet, B. Crémilleux, and O. Gandrillon. Constraint-based knowledge discovery from sage data. *In Silico Biology*, 8(0014), 2008.

[16] A. Knobbe, B. Crémilleux, J. Fürnkranz, and M. Scholz. From local patterns to global models : The lego approach to data mining. In *Int. Workshop LeGo co-located with ECML/PKDD'08*, pages 1–16, Antwerp, Belgium, 2008.

[17] Arno J. Knobbe and Eric K. Y. Ho. Pattern teams. In *PKDD*, pages 577–584, 2006.

[18] Laks V. S. Lakshmanan, Raymond T. Ng, Jiawei Han, and Alex Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *SIGMOD Conference*, pages 157–168. ACM Press, 1999.

[19] J.-P. Métivier, P. Boizumault, B. Crémilleux, M. Khiari, and S. Loudni. A constraint-based language for declarative pattern discovery. In *27th annual ACM Symposium on Applied Computing (SAC'12)*, pages 1–7, Riva del Garde (Trento), Italy, March 2012.

[20] R. T. Ng, V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *proceedings of ACM SIGMOD'98*, pages 13–24. ACM Press, 1998.

[21] Siegfried Nijssen and Tias Guns. Integrating constraint programming and itemset mining. In *ECML/PKDD (2)*, pages 467–482, 2010.

[22] Luc De Raedt and Albrecht Zimmermann. Constraint-based pattern set mining. In *SDM*. SIAM, 2007.

[23] Einoshin Suzuki. Undirected Discovery of Interesting Exception Rules. *IJPRAI*, 16(8) :1065–1086, 2002.

[24] Jérémie Vautard. *Modélisation et résolution de problèmes de décision et d'optimisation hiérarchiques en utilisant des contraintes quantifiées.* These, Université d'Orléans, April 2010.

[25] Dong Xin, Hong Cheng, Xifeng Yan, and Jiawei Han. Extracting redundancy-aware top-k patterns. In *KDD*, pages 444–453. ACM, 2006.

[26] X. Yin and J. Han. CPAR : classification based on predictive association rules. In *proceedings of the 2003 SIAM Int. Conf. on Data Mining (SDM'03)*, San Fransisco, CA, May 2003.

Calcul de solutions équilibrées Pareto-optimales : application au problème de gestion de dépendances logicielles

Daniel Le Berre, Emmanuel Lonca, Pierre Marquis, Anne Parrain

Université Lille Nord de France, F-59000 Lille, France
Université d'Artois, CRIL, F-62300 Lens, France
CNRS, UMR8188, F-62300 Lens, France
{nom}@cril.fr

Résumé

Notre travail se situe dans le cadre de la gestion de dépendances logicielles. Nous présentons dans cet article des améliorations de l'approche décrite dans [2] concernant le calcul de solutions équilibrées dans ce contexte. Plus précisément, nous nous intéressons ici au calcul de solutions équilibrées Pareto-optimales. Nous comparons plusieurs codages de ce problème à base de variables booléennes, dont la résolution est confiée au prouveur Sat4j, et un codage utilisant une variable entière, dont la résolution est confiée au prouveur CPLEX.

Abstract

Our work deals with software dependency management problems. In this article, we present some improvements of the approaches presented in [2] for computing balanced solutions in this context. More precisely, our aim is to to compute Pareto optimal balanced solutions. We compare some encodings using Boolean variables, implemented on top of the Sat4j solver, and an encoding using integer variables, implemented on top of the CPLEX solver.

1 Introduction

Le problème de gestion de dépendances logicielles concerne l'installation de programmes informatiques modulaires. Un module peut nécessiter la présence d'autres modules pour pouvoir fonctionner, il peut entrer en conflit avec certains autres modules, et parfois il peut recommander l'installation de modules spécifiques pour pouvoir être utilisé au meilleur de ses capacités. Il s'agit de déterminer, lorsqu'on veut installer ou mettre à jour un programme informatique, les modules à installer et les modules à enlever. Le problème de décision associé est NP-complet [10, 18]. Plusieurs approches pour résoudre ce problème ont été proposées ces dernières années dans le cadre des dépendances GNU/Linux [6].

Le problème de gestion des dépendances se modélise de manière naturelle à l'aide de variables booléennes, en associant à chaque paquetage une variable. La valeur de vérité de la variable correspond alors à l'état du paquetage dans le système : installé ou non. Les dépendances entre paquetages peuvent être représentées par des clauses. On obtient donc une instance du problème SAT, qui peut être résolue par l'un des nombreux prouveurs disponibles. Le détail du codage en logique propositionnelle que nous utilisons est décrit dans [1].

Les problèmes de gestion de dépendances sont généralement sous-contraints. Le problème n'est pas de trouver une solution, mais une solution qui a un sens pour la personne qui gère le système, par exemple, qui conduit à installer uniquement les paquetages nécessaires à l'ajout d'une fonctionnalité au système, ou bien qui limite les changements aux seules versions des logiciels installés. Un ensemble de six critères a été défini par le projet européen Mancoosi [4] pour évaluer les solutions obtenues par les outils de gestion de dépendances. Le problème de gestion de dépendances logicielles se réduit alors à un problème d'optimisation, en variables booléennes, multi-critères, sous contraintes. Il existe diverses approches pour traiter ces problèmes en variables booléennes (formalismes pseudo-booléen [1], MaxSat [1], formalisme ASP [5], programmation linéaire [13], ...)

Dans le cadre du projet Mancoosi, les critères étaient considérés de manière lexicographique. Cela implique un ordre total entre les critères, ce qui ne correspond pas

toujours aux préférences de l'utilisateur, car l'optimisation d'un critère peut dégrader fortement la valeur prise à l'optimum pour un autre critère moins préféré. Nous nous sommes donc intéressés au calcul de solutions « équilibrées » [2], c'est-à-dire aussi proches que possible de la valeur optimale pour chacun des critères. Nous avons utilisé pour cela la norme de Tchebycheff [17, 19]. Nous avons proposé des algorithmes de calcul de solution équilibrée applicables pour des problèmes codés sous forme de contraintes dans le cas multi-critère. Dans cet article, nous comparons dans un premier temps les codages existants avec un nouveau codage permettant de linéariser la fonction objectif multi-critère considérée. Dans un second temps, nous proposons une solution pour ne conserver que des solutions Pareto-optimales parmi les solutions équilibrées, propriété qui n'est pas assurée pour les solutions optimales selon la norme de Tchebycheff.

2 Norme de Tchebycheff

La norme de Tchebycheff permet de caractériser des solutions (ou alternatives réalisables) équilibrées. Elle se base sur la notion de **point idéal**, qui est le vecteur des coûts optimaux pour chaque critère, et qui ne représente bien souvent pas une alternative admissible.

Définition 1 (point idéal)**.** *Soit S l'ensemble des solutions, et soit $c_1, c_2, ..., c_n$ les fonctions de coût associées aux n critères. Le point idéal $(c_1^*, c_2^*, ..., c_n^*) \in \mathbb{N}^n$ est le vecteur tel que $\forall i \in \{1, 2, ..., n\}$, $c_i^* = min(\{c_i(s) \mid s \in S\})$.*

Pour un critère donné, la différence entre la valeur pour une solution et celle du point idéal exprime le **regret** de choisir cette solution.

Définition 2 (regret)**.** *Soit $x^* = (c_1^*, c_2^*, ..., c_n^*)$ le point idéal d'un problème à n critères et $c_1, c_2, ..., c_n$ les fonctions de coût associées à ces critères. Soit $s \in S$ une solution du problème. Le regret de s pour le $i^{ème}$ critère est la valeur $r_i(s) = c_i(s) - c_i^*$*

Nous pouvons maintenant définir la méthode **min-max regret**, qui préfère parmi plusieurs alternatives celles qui offrent le plus petit regret maximal sur les critères qui composent leur vecteur.

Définition 3 (min-max regret)**.** *Soient s et s' deux alternatives admissibles, et $r_1, r_2, ..., r_n$ les fonctions exprimant le regret sur les n critères. La solution s est préférée à la solution s' par la méthode du **min-max regret** si et seulement si :*

$$max(\{r_i(s) \mid i \in 1, ..., n\}) < max(\{r_i(s') \mid i \in 1, ..., n\}).$$

Une variante de cette méthode où les regrets associés à chaque critère sont pondérés est nommée **méthode de Tchebycheff**.

Définition 4 (norme et méthode de Tchebycheff)**.** *Soient s et s' deux alternatives admissibles, et $r_1, r_2, ..., r_n$ les fonctions exprimant le regret sur les n critères. Soit $W = (w_1, w_2, ..., w_n)$ un vecteur de poids entiers associés aux n critères. La **norme de Tchebycheff** de s, aussi appelée max-regret de s, est donnée par :*

$$U(s) = max(\{w_i \times r_i(s) \mid i \in 1, ..., n\}).$$

*La solution s est préférée à la solution s' par la **méthode de Tchebycheff** si et seulement si $U(s) < U(s')$*

3 Calcul de solutions équilibrées

Le problème de décision associé à la gestion de dépendances étant NP-complet, il est possible d'utiliser un codage en logique propositionnelle de ce problème pour le résoudre [8, 1, 11, 18]. Dans un premier temps, on peut ainsi calculer la valeur optimale pour chaque critère, et obtenir $(c_1^*, c_2^*, ..., c_n^*)$. Le calcul d'une solution équilibrée se fait ensuite par ajout de contraintes au problème initial. Nous ne détaillons pas ici l'ensemble C de contraintes utilisées pour représenter les dépendances entre logiciels, elles sont décrites dans [1].

3.1 Résolution successive de problèmes de satisfaction

Le premier codage que nous avons employé consiste simplement à ajouter itérativement des contraintes au problème de départ pour écarter les solutions dont la norme de Tchebycheff est supérieure à un certain seuil, jusqu'à obtenir une solution dont la norme de Tchebycheff est minimale. On notera s^* une telle solution optimale pour la norme de Tchebycheff.

La valeur optimale $U(s^*)$ se caractérise facilement. Si r est une valeur telle que $C \cup \{w_i \times r_i(s) \leq r \mid i \in 1, ..., n\}$ est satisfiable, et que $C \cup \{w_i \times r_i(s) \leq r - 1 \mid i \in 1, ..., n\}$ ne l'est pas, alors on a $r = U(s^*)$.

Comme chaque critère que nous considérons dans le cadre de la gestion de dépendances de paquets Linux est une fonction linéaire de variables booléennes [1], les contraintes $w_i \times r_i(s) \leq r$ $(\forall i \in \{1, ..., n\})$ peuvent s'exprimer sous forme de contraintes pseudo-booléennes : $w_i \times c_i(s) \leq r + w_i \times c_i^*$ $(\forall i \in \{1, ..., n\})$.

Le principal avantage de cette méthode est d'être purement basée sur des contraintes en variables booléennes, ce qui permet d'utiliser un vaste choix de prouveurs pour résoudre le problème d'optimisation.

3.2 Transformation en un problème d'optimisation linéaire

La norme de Tchebycheff étant une fonction dans laquelle apparaît un maximum, elle a le désavantage de ne

pas être linéaire. Cela a pour inconvénient de ne pas nous permettre d'utiliser les outils d'optimisation de fonctions linéaires existants, et de devoir faire appel à des méthodes *ad hoc*, comme celle présentée ci-dessus.

Cependant, dans le contexte d'un problème codé sous forme de contraintes en variables entières, cet obstacle peut être surmonté sans trop de difficultés. Nous avons vu que l'on pouvait majorer la valeur de la norme de Tchebycheff des solutions du problème en majorant les regrets de chacun des critères considérés. Or, notre but est de minimiser la norme de Tchebycheff de nos solutions, et cette norme est exprimée simplement à l'aide d'une variable entière r. Nous obtenons donc le problème d'optimisation suivant :

$$\begin{array}{rl} \min & r \\ \text{tel que} & C \cup \{w_i \times r_i(s) \leq r \mid i \in 1, ..., n\}. \end{array}$$

Nous avions initialement écarté cette approche car elle nécessite l'utilisation d'une variable entière alors que nous avions décidé de travailler dans le formalisme pseudo-booléen. Cependant, il est possible de représenter r sous sa forme binaire [3], $r = \sum_{i=0}^{m-1} 2^i \times b_i$; on obtient alors le problème booléen suivant :

$$\begin{array}{rl} \min & \sum_{i=0}^{m-1} 2^i \times b_i \\ \text{tel que} & C \cup \{w_i \times r_i(s) \leq \sum_{j=0}^{m-1} 2^j \times b_j \mid i \in 1, ..., n\}. \end{array}$$

4 Algorithmes

4.1 Optimisation par renforcement de contraintes

Un premier algorithme d'optimisation itératif consiste à partir d'une solution quelconque, et à restreindre l'espace de recherche tant que c'est possible. C'est une approche classique pour les moteurs booléens d'optimisation [15]. On exploite ici le fait que la norme de Tchebycheff prend ses valeurs dans $\mathbb{N}$ pour nos problèmes. Ainsi, au départ, nous recherchons simplement une solution au problème, indépendamment des valeurs qu'elle donne aux différents critères. Soit r la valeur de son max-regret. On renforce alors les contraintes du problème en fixant $r - 1$ comme majorant pour chacun des $w_i \times r_i(s)$. Si une solution existe, alors on itère le processus en utilisant le max-regret de cette nouvelle solution comme nouvelle valeur pour r. Dans le cas contraire, on conclut qu'il n'existe pas de solution s telle que $U(s) \in [0, r-1]$, et par conséquent les solutions qui donnent un max-regret de r sont optimales.

Cet algorithme est correct dans notre cas car les valeurs prises par la fonction objectif admettent toutes un prédécesseur et un nombre fini de minorants.

Le temps de calcul de cet algorithme est fortement dépendant de la valeur de la solution trouvée à la première itération. En effet, si cette valeur est n, l'algorithme effectuera au maximum n itérations. Une amélioration possible consiste à rechercher une solution relativement proche de la valeur optimale dès le début. Pour cela, nous avons utilisé une fonction linéaire qui approche la valeur de notre fonction objectif, et nous avons optimisé selon cette fonction linéaire. Le temps de calcul d'une solution optimale pour U par ajout de contraintes peut être réduit significativement lorsque la solution initiale est une « bonne » solution pour notre fonction d'approximation. Cela permet également de donner un temps limite d'exécution au moteur d'optimisation mono-critère, afin d'éviter de perdre trop de temps lorsque celui-ci se rapproche de la valeur optimale pour l'approximation. L'intérêt de cette approche est de pouvoir conserver les clauses apprises entre deux appels successifs au prouveur car celles-ci sont conséquences logiques de l'ensemble de contraintes initial (SAT incrémental).

Entrées: problème P
Entrées: fonction objectif multi-critère U
Sorties: solution réalisable s minimisant U
$s \leftarrow obtenirSolution(P)$;
tant que $s \neq UNSAT$ **faire**
  $r \leftarrow maxRegret(U, s)$;
  $meilleure \leftarrow s$;
  $contraintes \leftarrow P \cup cstrMaxRegret(U, r)$;
  $s \leftarrow obtenirSolution(contraintes)$;
fin
retourner *meilleure* ;

Algorithm 1: Renforcement de contraintes

4.2 Optimisation linéaire binaire

La deuxième approche se place dans le contexte de l'optimisation d'une fonction objectif « binaire » de la forme $\min 2^0 b_0 + ... + 2^{m-1} b_{m-1}$. Pour ce genre de problème, un algorithme spécifique a été proposé [3], assurant un maximum de m appels au solveur SAT pour effectuer l'optimisation. Cet algorithme se base sur une optimisation dichotomique. En effet, la fonction objectif considérée peut prendre un maximum de 2^m valeurs. Afin de déterminer s'il existe une solution dont le coût est inférieur à 2^{m-1}, il suffit de chercher s'il existe une solution en fixant la variable b_{m-1} à la valeur 0. S'il existe une telle solution, alors on conserve cette valeur de vérité ; dans le cas contraire, on la fixe à la valeur 1. On effectue cette opération m fois, soit successivement pour les variables $b_{m-1}, ...b_1, ...b_0$.

Il est par ailleurs possible de sauter des étapes. Imaginons que le solveur trouve une solution de coût $c = \sum_{i=0}^{k} 2^i b_i + K$ pour un k entre $m-1$ et 0. Alors il est possible de conserver la valeur de vérité des b_i pour i de k à 0, tant que $b_i = 0$.

4.3 Optimisation par relaxation de contraintes

Il s'agit de l'approche duale de la première : on considère au départ une valeur de r qui minore $U(s^*)$. Si $C \cup \{w_i r_i(s) \leq r \mid i \in 1, ..., n\}$ possède une solution, alors cette solution est optimale. Dans le cas restant, on « relâche » les contraintes imposées en substituant $\{w_i r_i(s) \leq r \mid i \in 1, ..., n\}$ par $\{w_i r_i(s) \leq r + 1 \mid i \in 1, ..., n\}$ de manière à tester la valeur de r immédiatement supérieure à la précédente. On itère ce processus. Lorsqu'on trouve une solution, elle est nécessairement optimale (sinon on aurait trouvé une solution lors d'une itération passée). Cette approche rencontre un vif succès dans le cadre du problème d'optimisation MAXSAT lorsqu'elle est couplée à la détection de noyau incohérent [16]. Voir l'algorithme 2.

Entrées: problème P
Entrées: fonction objectif multi-critère U
Sorties: solution réalisable s minimisant U
$r \leftarrow 0$;
$contraintes \leftarrow P \cup cstrMaxRegret(U, r)$;
$s \leftarrow obtenirSolution(contraintes)$;
tant que $s = UNSAT$ **faire**
  $r \leftarrow r + 1$;
  $contraintes \leftarrow P \cup cstrMaxRegret(U, r)$;
  $s \leftarrow obtenirSolution(contraintes)$;
fin
retourner s ;

Algorithm 2: Relaxation de contraintes

Pour que cet algorithme soit correct, il faut disposer d'une valeur qui minore $U(s^*)$ (c'est le point de départ et dans notre cadre la valeur 0 convient), et faire varier r de façon à garantir que $U(s^*)$ sera atteint après un nombre fini d'itérations. Cela est le cas quand les fonctions de coût c_i sont à valeur dans $\mathbb{N}$ et que l'incrément choisi pour r est 1. L'intérêt de cette approche est de nécessiter exactement $U(s^*)$ appels au prouveur pour trouver une solution optimale : c'est intéressant pour les petites valeurs de $U(s^*)$. Contrairement aux cas précédents, il n'est pas possible de conserver les clauses apprises d'un appel à l'autre, ce qui exclut une approche collaborative par dichotomie.

5 Vers des solutions Pareto-optimales

Un des inconvénients majeurs dû à l'utilisation de la norme de Tchebycheff comme fonction d'agrégation de critères est que les solutions retournées ne sont pas nécessairement Pareto-optimales. En effet, considérons un problème bi-critère ayant deux solutions s_1 et s_2 optimales pour la norme de Tchebycheff. Supposons que ces solutions aient respectivement pour vecteurs de regrets $(2, 2)$ et $(2, 1)$. Les deux solutions ont effectivement un regret maximal de 2, mais on remarque que le deuxième vecteur domine le premier au sens de Pareto. Il est alors naturel de choisir la solution s_2. Or, aucun des algorithmes présentés plus haut ne peut assurer quelle sera la solution choisie. Nous proposons maintenant une méthode capable de pallier cet inconvénient.

L'obtention de solutions Pareto-optimales peut être garanti par un post-traitement, sous la forme d'une nouvelle fonction à optimiser. Cela fournit une méthode générique utilisable indépendamment de l'algorithme employé pour obtenir des solutions équilibrées. L'éventuel inconvénient pouvant survenir est une perte d'efficacité : le temps nécessaire au calcul de telles solutions serait peut-être moins important en modifiant directement nos algorithmes ou nos codages, et constitue une perspective de travail.

Afin de choisir une solution Pareto-optimale parmi les solutions Tchebycheff-optimales, on ajoute la contrainte suivante à notre problème (r est le max-regret minimal) :

$$C \cup \{w_i \times r_i(s) \leq r \mid i \in 1, ..., n\}.$$

Une fois ces contraintes ajoutées, il suffit de minimiser la fonction $\sum_{i=0}^{n} c_i(s)$ pour garantir la Pareto-optimalité.

Il est possible de combiner lexicographiquement la fonction linéaire présentée en section 3.2 et cette fonction d'optimisation assurant la Pareto-optimalité en une seule fonction linéaire. Cela permettrait d'obtenir un problème d'optimisation mono-critère. Cependant, cela se ferait au prix d'une perte de la prise en compte spécifique de la fonction linéaire « binaire ».

6 Expérimentations et résultats

Nous avons implémenté les différentes approches présentées dans une version modifiée de p2cudf [1], une adaptation du logiciel p2 [8] capable de résoudre des problème de gestion de dépendances pour GNU/Linux, basé sur Sat4j [7], une bibliothèque Java de prouveurs pour résoudre des problèmes de décision ou d'optimisation en variables booléennes. Nos expérimentations ont été réalisées sur un Quad-core Intel XEON X5550 avec 32Go de mémoire. Nous avons testé nos algorithmes sur des instances issues des compétitions *misc* [1] organisées dans le cadre du projet mancoosi. Nous nous sommes intéressés à trois séries d'instances : les *dudf-real* de la compétition *misc2011*, et les *9orless* et *10orplus* de la compétition *misc-live*. Il s'agit de problèmes réels d'installation de paquetages pour la distribution GNU/Linux Debian. Pour chaque instance, un temps limite d'exécution de dix minutes a été imposé.

Les tests ont été effectués sur deux (*removed*, *versionchanged*) puis trois (*removed*, *versionchanged*, *new*) critères à minimiser. *removed* correspond au nombre de pa-

1. `http://www.mancoosi.org/misc/`

quetages qui ont été enlevés du système, *versionchanged*[2] correspond au nombre de logiciels installés qui ont simplement changé de version et *new* correspond au nombre de logiciels non installés à installer.

	défaut	Pareto-opt.
renforcement	216	208
	15439s	18500s
linéaire binaire	206	203
	16085s	19381s
relaxation	214	206
	13220s	15593s
CPLEX	254	254
	21608s	22043s

FIGURE 1 – Récapitulatif des résultats des expérimentations (total de 254 instances, et temps de calcul cumulé de ces instances en secondes)

La figure 1 présente le nombre de problèmes résolus (sur 254) et le temps cumulé nécessaire pour résoudre ces problèmes. Nous considérons qu'un problème est résolu quand notre logiciel retourne une solution prouvée optimale par Sat4j. Il arrive très souvent que Sat4j trouve une solution optimale mais ne soit pas en mesure de prouver cette optimalité. C'est la raison pour laquelle l'outil p2cudf a obtenu de bons résultats lors de la compétition *misc2011*[3] car la preuve de l'optimalité n'est pas prise en compte. C'est un problème connu de Sat4j sur lequel nous travaillons. On peut noter tout d'abord que des trois codages que nous avons comparés, les deux basés sur l'ajout successif de contraintes donnent des résultats comparables. L'approche par linéarisation binaire est légèrement moins performante. En ce qui concerne la recherche de solutions Pareto-optimales, on remarque que le temps de calcul nécessaire à cette étape supplémentaire n'est pas négligeable. Avec le temps de calcul limité à dix minutes, les différentes versions de nos solveurs n'arrivent pas nécessairement à déterminer une solution Pareto-optimale alors qu'ils réussissent à calculer une solution Tchebycheff-optimale. Au niveau de la qualité des solutions, en revanche, on remarque une amélioration, surtout lorsqu'on considère les instances les plus difficiles (*10orplus*) avec trois critères. Voir la figure 2.

Nous avons aussi comparé nos approches en utilisant le solveur CPLEX 12 à la place de Sat4j. Pour ce solveur, nous avons utilisé le codage présenté en section 4.2 qui ne requiert pas de décomposition en puissances de deux la variable de la fonction objectif. En dépit du fait que nous communiquons avec CPLEX par ficihers et que cela a un coût non négligeable lors de l'échange d'informations avec

2. Le critère *changed* de Mancoosi correspond à *removed* + *versionchanged* + *new*. Nous avons introduit le critère *versionchanged* pour utiliser des critères indépendants.

3. `http://www.mancoosi.org/misc-2011/results/`

instance	défaut	Pareto	CPLEX pareto
epiphany-browser	75,23,18	75,23,17	75,23,17
	$751s$	$753s$	$132s$
evolution-dev	55,15,11	55,14,8	55,13,9
	$150s$	$224s$	$127s$
gnochm	39,4,5	39,3,4	38,4,4
	$101s$	$131s$	$121s$
gnome-panel-data	37,4,1	37,3,0	37,3,0
	$102s$	$136s$	$122s$

FIGURE 2 – Exemple de l'amélioration apportée par la recherche de solutions Pareto-optimales

p2cudf, CPLEX réussit à résoudre toutes nos instances, et rapidement pour la plupart d'entre elles. Ceci est cohérent avec les résultats obtenus par le prouveur UNSA lors de *misc2010*.

7 Conclusions et perspectives

Dans cet article, nous avons dans un premier temps comparé plusieurs codages et plusieurs algorithmes capables de résoudre des problèmes d'optimisation multi-critère sous contraintes pseudo-booléennes. Nous avons constaté que le codage pour lequel l'optimisation passe par plusieurs itérations « ajout de contraintes – test de cohérence » est plus efficace sur nos problèmes, quand on utilise notre solveur.

En revanche, nous n'avons pas pu départager les deux algorithmes que nous avons comparés. En effet, l'algorithme d'optimisation par renforcement de contraintes arrive à résoudre quelques instances de plus dans le temps imparti que celui qui optimise par relaxation de contraintes, mais il est généralement plus lent pour résoudre les instances que le solveur utilisant la relaxation lorsque ce dernier trouve une solution optimale.

Nous avons cherché à obtenir des solutions Pareto-optimales, ce qui n'est pas une propriété assurée par l'optimalité au sens de la norme de Tchebycheff. Pour cela, nous avons ajouté une étape de post-traitement qui permet d'obtenir une solution Tchebycheff-optimale Pareto-optimale. Cette étape se résume à une optimisation mono-critère : la minimisation de la somme des coûts associés à nos critères.

Nous avons aussi comparé nos approches en utilisant le solveur CPLEX à la place de Sat4j. CPLEX s'est révélé beaucoup plus performant. Cela ne remet pas en forcément en cause l'approche en variables booléennes. D'autres prouveurs en variables booléennes se rélèvent plus efficaces que Sat4j sur ces problèmes de gestion de dépendances (Clasp [5] et WBO [12] par exemple, voir les résultats de *misc2010* et *misc2011*). Nous utilisons Sat4j car nous maîtrisons complètement son fonctionnement, et parce que nous espérons aussi améliorer ses performances sur ce type de problèmes.

Une piste pour des travaux futurs est de comparer nos résultats avec d'autres méthodes d'optimisations multicritère, comme les OWR [14], qui ont l'avantage de fournir des solutions Pareto-optimales. Avec les OWR, il est possible de garantir la Tchebycheff-optimalité en plus de la Pareto-optimalité en considérant le poids associé au max-regret comme très important, voire même de se ramener à un leximax si les écarts entre les différents poids sont tous très grands.

Références

[1] Josep Argelich, Daniel Le Berre, Inês Lynce, João P. Marques Silva, and Pascal Rapicault. Solving linux upgradeability problems using boolean optimization. In Lynce and Treinen [9], pages 11–22.

[2] Daniel Le Berre, Emmanuel Lonca, Pierre Marquis, and Anne Parrain. Optimisation multicritère pour la gestion de dépendances logicielles : utilisation de la norme de tchebycheff. In *Actes de la conférence RFIA 2012*. RFIA, 2012.

[3] Michael Codish, Samir Genaim, and Peter J. Stuckey. A declarative encoding of telecommunications feature subscription in sat. In António Porto and Francisco Javier López-Fraguas, editors, *PPDP*, pages 255–266. ACM, 2009.

[4] R. Di Cosmo and S. Cousin. Project presentation. Technical report, Mancoosi (Managing the Complexity of the Open Source Infrastructure), 2008.

[5] Martin Gebser, Roland Kaminski, and Torsten Schaub. aspcud : A linux package configuration tool based on answer set programming. In Conrad Drescher, Inês Lynce, and Ralf Treinen, editors, *LoCoCo*, volume 65 of *EPTCS*, pages 12–25, 2011.

[6] G. Guttierez, M. Janota, I. Lynce, O. Lhomme, V. Manquinho, J. Marques-Silva, and C. Michel. Final version of the optimizations algorithms and tools. Technical report, Mancoosi (Managing the Complexity of the Open Source Infrastructure), 2011.

[7] D. Le Berre and A. Parrain. The sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7 :59–64, 2010.

[8] D. Le Berre and P. Rapicault. Dependency management for the eclipse ecosystem : eclipse p2, metadata and resolution. In *Proceedings of the 1st international workshop on Open component ecosystems*, pages 21–30. ACM, 2009.

[9] Inês Lynce and Ralf Treinen, editors. *Proceedings First International Workshop on Logics for Component Configuration*, volume 29 of *EPTCS*, 2010.

[10] F. Mancinelli, J. Boender, R. Di Cosmo, J. Vouillon, B. Durak, X. Leroy, and R. Treinen. Managing the complexity of large free and open source package-based software distributions. In *ASE*, pages 199–208. IEEE Computer Society, 2006.

[11] F. Mancinelli, J. Boender, R. Di Cosmo, J. Vouillon, et al. Managing the complexity of large free and open source package-based software distributions. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 199–208. IEEE Computer Society, 2006.

[12] Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In Oliver Kullmann, editor, *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009.

[13] Claude Michel and Michel Rueher. Handling software upgradeability problems with milp solvers. In Lynce and Treinen [9], pages 1–10.

[14] Wlodzimierz Ogryczak, Patrice Perny, and Paul Weng. On Minimizing Ordered Weighted Regrets in Multiobjective Markov Decision Processes. In Ronen I. Brafman, Fred S. Roberts, and Alexis Tsoukiàs, editors, *ADT*, volume 6992 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2011.

[15] Olivier Roussel and Vaso Manquinho. Pseudo-boolean and cardinality constraints. In *Handbook of Satisfiability*, pages 695–733. IOS Press, February 2009.

[16] João P. Marques Silva. Minimal unsatisfiability : Models, algorithms and applications (invited paper). In *ISMVL*, pages 9–14. IEEE Computer Society, 2010.

[17] R.E. Steuer and E.U. Choo. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26(3) :326–344, 1983.

[18] Tommi Syrjänen. A rule-based formal model for software configuration. Research Report A55, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, December 1999.

[19] A.P. Wierzbicki. On the completeness and constructiveness of parametric characterizations to vector optimization problems. *OR Spectrum*, 8(2) :73–87, 1986.

Un Solveur Léger Efficace pour Interroger le Web Sémantique *

V. le Clément de Saint-Marcq[1,2,3] **Y. Deville**[1] **C. Solnon**[2,4] **P.-A. Champin**[2,3]

[1] Université catholique de Louvain, ICTEAM institute, B-1348 Louvain-la-Neuve (Belgium)
[2] Université de Lyon, LIRIS, CNRS UMR5205, 69622 Villeurbanne (France)
[3] Université Lyon 1, 69622 Villeurbanne (France)
[4] INSA Lyon, 69622 Villeurbanne (France)
{vianney.leclement,yves.deville}@uclouvain.be christine.solnon@liris.cnrs.fr

Résumé

Le Web Sémantique vise à construire des bases de données inter-domaines et distribuées à travers l'internet. SPARQL est un langage de requêtes standard pour ces bases de données. L'évaluation de telles requêtes est cependant NP-difficile. Nous modélisons les requêtes SPARQL de manière déclarative, au moyen de CSPs. Une sémantique opérationnelle CP est proposée. Elle peut être utilisée pour une implémentation directe dans des solveurs CP existants. Pour traiter des bases de données de grande taille, nous introduisons un solveur spécialisé, léger et efficace, Castor. Les benchmarks montrent la faisabilité et l'efficacité de l'approche.

1 Introduction

L'Internet est devenu le moyen privilégié de recherche d'information dans la vie de tous les jours. Bien que l'information disponible en abondance sur le Web soit de plus en plus accessible pour les utilisateurs humains, les ordinateurs ont encore du mal à la comprendre. Les développeurs doivent s'appuyer sur des techniques d'apprentissage machine probabilistes [5] ou des APIs dédiées à certains sites (par exemple, Google API), ou recourir à l'écriture d'un analyseur spécialisé devant être mis à jour à chaque changement de mise en page.

Le Web Sémantique est une initiative du World Wide Web Consortium (W3C) pour permettre aux sites de publier des données lisibles par la machine à côté des documents destinés à l'être humain. La fusion de toutes les données publiées sur le Web Sémantique résulte en une grande base de données globale. Ce caractère global du Web sémantique implique une structure beaucoup plus souple que les bases de données relationnelles traditionnelles. Cette structure permet de stocker des données non liées, mais rend l'interrogation de la base plus difficile. SPARQL [16] est un langage de requête pour le Web Sémantique qui a été standardisé par le W3C. Évaluer les requêtes SPARQL est connu pour être NP-difficile [15].

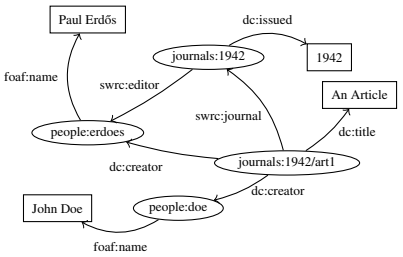
Le modèle d'exécution des moteurs SPARQL actuels (par exemple, Sesame [4], 4store [9] ou Virtuoso [7]) est basé sur l'algèbre relationnel. Une requête est subdivisée en sous-requêtes qui sont calculées séparément. Les ensembles de réponse sont ensuite fusionnés. Les filtres additionnels spécifiés par l'utilisateur sont souvent traités après ces opérations de jointure. La Programmation par Contraintes (CP), par contre, est en mesure d'exploiter les filtres comme des contraintes lors de la recherche. Un moteur de recherche basé sur les contraintes est donc bien adapté pour le Web Sémantique.

Contributions. Notre première contribution est un modèle déclaratif basé sur le formalisme des problèmes de satisfaction de contraintes (CSP) et une sémantique opérationnelle basée sur la programmation par contraintes (CP) pour résoudre des requêtes SPARQL. Les solveurs CP existants ne sont toutefois pas conçus pour traiter les immenses domaines en lien avec les bases de données du Web sémantique. La deuxième contribution de ce travail est un solveur spécialisé léger, appelée Castor, pour l'exécution de requêtes SPARQL. Sur des benchmarks standards, Castor est compétitif avec les moteurs existants et améliore l'état de l'art sur des requêtes complexes.

*Ce papier est une version étendue de travaux publiés à CP 2011 et ESWC 2012.

```
people:erdoes foaf:name   "Paul Erdős" .
people:doe    foaf:name   "John Doe" .
journals:1942 dc:issued   1942 .
journals:1942 swrc:editor people:erdoes .
journals:1942/art1 dc:title   "An Article" .
journals:1942/art1 dc:creator people:erdoes .
journals:1942/art1 dc:creator people:doe .
```

(a) Ensemble de triplets



(b) Représentation par un graphe

FIGURE 1 – Exemple de base RDF représentant un journal fictif édité par Paul Erdős et un article de ce journal écrit par Erdős et Doe. Ici, `people:erdoes` et `foaf:name` sont des URIs alors que `"Paul Erdős"` est un littéral.

Plan. La section suivante explique comment les données sont représentées dans le Web Sémantique et la façon d'interroger les données. Les section 3 et 4 montrent respectivement le modèle déclaratif et la sémantique opérationnelle pour résoudre les requêtes. La section 5 présente notre solveur léger implémentant la sémantique opérationnelle. La section 6 évalue la faisabilité et l'efficacité de notre approche au travers d'un benchmark standard.

2 Le Web Sémantique et le Langage de Requêtes SPARQL

Le *Resource Description Framework* (RDF) [10] permet de modéliser la connaissance comme un ensemble de triplets $(\text{sujet}, \text{prédicat}, \text{objet})$. Ces triplets expriment des relations, décrites par les prédicats, entre sujets et objets. Les trois éléments d'un triplet peuvent être des ressources arbitraires identifiées par des *Uniform Resource Identifiers* (URI) [1]. Les objets peuvent également être des valeurs littérales, telles que des chaînes de caractères, des nombres, des dates ou des données spécialisées. Une base de données RDF peut être représentée par un multigraphe étiqueté orienté comme le montre la figure 1.

SPARQL [16] est un langage de requête pour RDF. Une requête basique est un ensemble de motifs de triplets, à savoir des triplets où les éléments peuvent être remplacés par des variables. Les requêtes basiques peuvent être assemblées dans des requêtes composées avec éventuellement des parties optionnelles ou alternatives. Les filtres ajoutent des contraintes sur les variables. Une solution d'une requête est une affectation de variables à des URIs ou littéraux figurant dans la base de données. La substitution des variables dans la requête par leurs valeurs attribuées dans la solution donne un sous-ensemble de la base de données. Une requête SPARQL peut également définir un sous-ensemble de variables à renvoyer, un ordre de tri, etc. mais, ceci n'étant pas pertinent pour ce papier, a été omis.

Plus formellement, soient U, L et V des ensembles infinis disjoints deux-à-deux représentant respectivement les URIs, les littéraux et les variables. Une instance du problème SPARQL est définie par une paire (S, Q) telle que $S \subset U \times U \times (U \cup L)$ est un ensemble fini de triplets correspondant à la base de données, et Q est une requête. La syntaxe des requêtes est définie récursivement comme suit [2]. La sémantique sera définie dans la section suivante.

- Une requête basique est un ensemble de motifs de triplets (s, p, o) tels que $s, p \in U \cup V$ et $o \in U \cup L \cup V$. La différence avec les bases RDF est que nous pouvons avoir des variables à la place des URI et des littéraux.
- Soient Q_1 et Q_2 des requêtes. Q_1 . Q_2, Q_1 `OPTIONAL` Q_2 et Q_1 `UNION` Q_2 sont des requêtes composées.
- Soient Q une requête et c une contrainte de sorte que chaque variable de c apparaît au moins une fois dans Q. Q `FILTER` c est une requête contrainte. Le langage d'expressions de SPARQL utilisé pour définir c comprend des opérateurs arithmétiques, booléens, et de comparaison, des expressions régulières pour les littéraux et certains opérateurs spécifiques à RDF.

Étant donnée une base de données S, U_S et L_S dénotent respectivement les ensembles d'URIs et de littéraux qui apparaissent dans S. Étant donnée une requête Q, $\text{vars}(Q)$ dénote l'ensemble des variables apparaissant dans Q.

3 Un Modèle Déclaratif CSP des Requêtes SPARQL

Une solution à une instance de problème SPARQL (S, Q) est une affectation σ de variables de Q à des valeurs de $U_S \cup L_S$, en d'autres termes un ensemble de paires variable/valeur. Étant données une solution σ et une requête Q, $\sigma(Q)$ dénote la requête obtenue en remplaçant chaque occurrence d'une variable affectée dans σ par sa valeur. Le but est de trouver toutes les solutions. On note $\text{sol}(S, Q)$ l'ensemble de toutes les solutions à (S, Q).

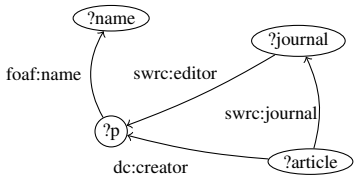
Contrairement aux CSPs classiques, une solution σ ne doit pas nécessairement couvrir toutes les variables apparaissant dans Q. Par exemple, si une variable x apparaît uni-

1. Plus précisément, RDF fait usage de *URI references*, mais la différence n'est pas pertinente au présent papier. La spécification permet aussi de remplacer les sujets et les objets par des nœuds vierges, c'est à dire des ressources sans identificateur. Sans perte de généralité, les nœuds vierges seront considérés comme URIs réguliers dans cet article.

2. Par souci de clarté, nous apportons quelques simplifications au langage. Ces hypothèses n'altèrent pas l'expressivité de SPARQL.

```
SELECT *
WHERE {
  ?p foaf:name ?name .
  ?journal swrc:editor ?p .
  ?article swrc:journal ?journal .
  ?article dc:creator ?p .
}
```

(a) Requête SPARQL



(b) Graphe motif associé

FIGURE 2 – Exemple de requête basique recherchant les éditeurs de journaux ayant publié un article dans le même journal. Les variables sont préfixées d'un point d'interrogation, par exemple `?name`. L'exécution de la requête sur la base de données de la figure 1 donne la solution unique {(p,people:erdoes), (name,"Paul Erdős"), (journal,journals:1942), (article,journals:1942/art1)}.

quement dans une partie optionnelle qui n'a pas été trouvée dans une solution s, x n'apparaîtra pas dans la solution σ. Ces variables sont dites non liées.

Dans cette section, nous définissons l'ensemble des solutions d'une instance de problème SPARQL au moyen de CSPs, donnant ainsi une sémantique dénotationnelle des requêtes SPARQL. Notez que, ce faisant, nous transformons un langage déclaratif, SPARQL, en un autre basé sur les CSPs qui peuvent être résolus par des solveurs existants.

3.1 Requêtes Basiques

Une requête basique BQ est un ensemble de motifs de triplets (s,p,o). Dans cette forme simple, une affectation σ est une solution si $\sigma(BQ) \subseteq S$.

Le problème SPARQL (S,BQ) peut être considéré comme un problème d'appariement de graphes d'un graphe requête associé à BQ à un graphe cible associé à S [3], comme illustré dans la figure 2. Cependant, même cette forme basique de requête est plus générale que les appariements de graphes classiques, tels que l'homomorphisme de graphes ou l'isomorphisme de sous-graphes. Les variables sur les arcs (les prédicats) peuvent imposer des relations additionnelles entre différents arcs. Ce problème est donc déjà NP-difficile.

Nous définissons formellement l'ensemble $\mathrm{sol}(S,BQ)$ comme les solutions du CSP (X,D,C) tel que

– $X = \mathrm{vars}(BQ)$,
– toutes les variables ont le même domaine, contenant tous les URIs et littéraux de S, c'est à dire $\forall x \in X, D(x) = U_S \cup L_S$,
– les contraintes assurent que chaque triplet de la requête appartient à la base de données, c'est à dire

$$C = \{\mathrm{Member}\big((s,p,o),S\big) \mid (s,p,o) \in BQ\}$$

où Member est la contrainte d'appartenance à un ensemble.

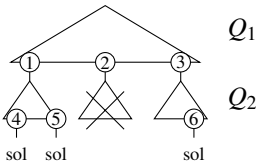
3.2 Requêtes Composées

Les requêtes plus avancées, comportant par exemple des parties optionnelles, ne peuvent être traduites directement en CSP. En effet, certaines requêtes s'appuient sur la non-satisfiabilité d'une sous-requête, qui est coNP-difficile. Les CSP ne peuvent modéliser que des problèmes NP.

Q_1 . Q_2. Deux requêtes peuvent être concaténées avec le symbole de jointure ou de concaténation (.). L'ensemble solution de la concaténation est le produit cartésien des ensembles solution des deux requêtes. Un tel produit cartésien est obtenu en fusionnant chaque paire de solutions affectant les mêmes valeurs aux variables communes. Notez que l'opérateur est commutatif, c.à.d. Q_1 . Q_2 est équivalent à Q_2 . Q_1. L'ensemble des solutions est défini par :

$$\mathrm{sol}(S, Q_1 \,.\, Q_2) = \{\, \sigma_1 \cup \sigma_2 \,|\, \sigma_1 \in \mathrm{sol}(S,Q_1), \; \sigma_2 \in \mathrm{sol}(S, \sigma_1(Q_2)) \,\} \;.$$

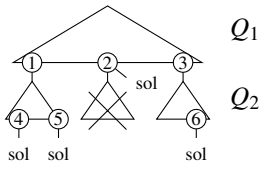
La figure ci-dessous représente un exemple. Un triangle représente un arbre de recherche d'une sous-requête. Les cercles au bas d'un triangle sont les solutions de la sous-requête. Les cercles 1, 2 et 3 représentent $\mathrm{sol}(S,Q_1)$. La solution 1 est étendue en les solutions 4 et 5 dans l'arbre de recherche de $\mathrm{sol}(S,\sigma_1(Q_2))$. Les solutions 4, 5 et 6 sont les solutions de la concaténation. Si Q_1 et Q_2 sont toutes deux des requêtes basiques, nous pouvons calculer la concaténation plus efficacement en fusionnant les deux ensembles de motifs de triplets et résoudre la requête basique résultante comme expliqué dans la section 3.1.



Q_1 `OPTIONAL` Q_2. L'opérateur `OPTIONAL` résout la sous-requête de gauche Q_1 et *essaie* de résoudre la sous-requête de droite Q_2. Si une solution de Q_1 ne peut être étendue en une solution de Q_1 . Q_2, alors cette solution de Q_1 devient

aussi une solution de la requête. Plus formellement,

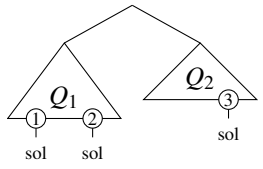
$$\text{sol}(S, Q_1 \texttt{ OPTIONAL } Q_2) = \text{sol}(S, Q_1 \,.\, Q_2) \cup \{\, \sigma \in \text{sol}(S, Q_1) \mid \text{sol}(S, \sigma(Q_2)) = \varnothing \,\} \quad .$$



Par rapport à l'exemple de l'opérateur de concaténation, le cercle 2 dans la figure devient une solution de la requête composée. Le test d'incohérence rend la recherche difficile. En effet, dans le cas simple, Q_2 est une requête basique et est donc modélisée par un CSP. Toutefois, puisque la vérification de la cohérence d'un CSP est NP-difficile, la vérification de son incohérence est coNP-difficile. Afin de garantir la compositionnalité de la sémantique, nous imposons que si Q_1 `OPTIONAL` Q_2 est une sous-requête d'une requête Q, les variables apparaissant dans Q_2 mais pas dans Q_1 ($\text{vars}(Q_2) \setminus \text{vars}(Q_1)$) n'apparaissent pas ailleurs dans Q. Cette condition n'altère pas l'expressivité du langage [1].

Q_1 `UNION` Q_2. Les disjonctions sont introduites par l'opérateur `UNION`. L'ensemble solution de l'union de deux requêtes est l'union des ensembles solution des deux requêtes. Les solutions des deux requêtes sont calculées séparément :

$$\text{sol}(S, Q_1 \texttt{ UNION } Q_2) = \text{sol}(S, Q_1) \cup \text{sol}(S, Q_2) \quad .$$



3.3 Filtres

L'opérateur `FILTER` retire les solutions de Q ne satisfaisant pas la contrainte c, à savoir :

$$\text{sol}(S, Q \texttt{ FILTER } c) = \{\, \sigma \in \text{sol}(S, Q) \mid c(\sigma) \,\} \quad ,$$

où $c(\sigma)$ est vrai si c est satisfait par σ. La référence SPARQL [16] définit la sémantique des contraintes, également en cas de variables non liées.

L'opérateur `FILTER` peut être utilisé a posteriori, pour enlever les solutions qui ne satisfont pas à certaines contraintes. C'est ce qui est fait habituellement par les moteurs SPARQL existants. Cependant, ces contraintes peuvent aussi être utilisées pendant le processus de recherche afin d'élaguer l'arbre de recherche. Un des buts de cet article est d'étudier le bénéfice de l'utilisation de CP, qui exploite activement les contraintes pour réduire l'espace de recherche, pour résoudre les requêtes SPARQL.

Lorsque l'opérateur `FILTER` est appliqué directement à une requête basique BQ, les contraintes peuvent être simplement ajoutées à l'ensemble des contraintes d'appartenance associé à la requête, c.à.d. $\text{sol}(S, BQ \texttt{ FILTER } c)$ est égal à l'ensemble solution du CSP $(X, D, C \cup \{c\})$, où (X, D, C) est le CSP associé à (S, BQ). Bien sûr, trouver toutes les solutions du CSP $(X, D, C \cup c)$ est généralement plus rapide que trouver toutes les solutions pour (X, D, C), et puis supprimer celles qui ne satisfont pas c.

Les filtres appliqués sur des requêtes composés peuvent parfois être *poussés* dans les sous-requêtes [18]. Par exemple $(Q_1 \texttt{ UNION } Q_2) \texttt{ FILTER } c$ peut être réécrit sous la forme $(Q_1 \texttt{ FILTER } c) \texttt{ UNION } (Q_2 \texttt{ FILTER } c)$. De telles optimisations de requête sont courantes dans les moteurs de base de données.

4 Une Modélisation CP Opérationnelle de requêtes SPARQL

La sémantique dénotationnelle de SPARQL peut être transformée en une sémantique opérationnelle avec des solveurs CP conventionnels à condition que ceux-ci permettent de poster des contraintes lors de la recherche. Des exemples de tels solveurs sont Comet [6] ou Gecode [8]. Nous détaillons la sémantique opérationnelle de requêtes SPARQL, c.à.d. comment l'ensemble $\text{sol}(S, Q)$ est calculé. Ce modèle peut être utilisé pour une implémentation directe dans les solveurs existants.

Pour exécuter une requête Q dans une base de données S, nous définissons un tableau global de variables CP $X = \text{vars}(Q)$. Le domaine initial de chaque variable $x \in X$ est $D(x) = U_S \cup L_S$. L'ensemble des contraintes C est initialement vide. Pour expliquer le postage des contraintes et la recherche, nous utilisons Comet comme notation. Le code suivant résout la requête Q :

```
solveall<cp> {
} using {
  sol(Q);
  output(); // affiche la solution
}
```

Le premier bloc (vide) poste les contraintes, le second décrit la recherche. La fonction `sol(Q)` sera définie pour chaque type de requête. Elle poste des contraintes et introduit des points de choix. Les points de choix sont soit explicites avec les mots-clés `try`, soit implicites lors de l'étiquetage des variables avec `label`. Quand un échec est rencontré, soit explicitement avec `cp.fail()`, soit implicitement lors de la propagation d'une contrainte, la recherche retourne au dernier point de choix et reprend l'exécution dans l'autre branche. Un retour en arrière se produit également après avoir affiché une solution à la fin du bloc `using`

```
function sol(BQ FILTER c) {
  forall((s,p,o) in BQ)
    cp.post(Member((s,p,o),S));
  cp.post(c);
  label(vars(BQ));
}
```

(a) Requête basique avec filtre

```
function sol(Q FILTER c) {
  sol(Q);
  if( ! c )
    cp.fail();
}
```

(b) Requête composée avec filtre

FIGURE 3 – Les filtres appliquées aux requêtes basiques sont postées comme des contraintes. Dans tous les autres cas, ils sont vérifiés après avoir résolu la sous-requête.

```
function
sol(Q1 . Q2) {
  sol(Q1);
  sol(Q2);
}
```

(a) Concaténation

```
function
sol(Q1 UNION Q2) {
  try<cp> {
    sol(Q1);
  }|{
    sol(Q2);
  }
}
```

(c) Union

```
function
sol(Q1 OPTIONAL Q2) {
  sol(Q1);
  Boolean cons(false);
  try<cp> {
    sol(Q2);
    cons := true;
  }|{
    if(cons)
      cp.fail();
  }
}
```

(b) Optional

FIGURE 4 – Les requêtes composées sont résolues récursivement.

pour chercher les autres solutions. Nous supposons une recherche en profondeur d'abord, développant les branches de gauche à droite.

Comme nous n'étiquetons pas toutes les variables dans chaque branche, les domaines de certaines variables peuvent encore être intacts lors de l'affichage d'une solution. Ces variables sont considérées comme non liées et ne sont pas incluses dans la solution. En effet, nous étiquetons toujours toutes les variables d'une requête basique. Les variables non liées n'apparaissent pas dans les requêtes basique le long d'une branche, en raison de disjonctions introduites par `UNION` ou de sous-requêtes optionnelles incohérentes. Aucune contrainte n'est postée sur ces variables. Leurs domaines ne sont donc pas réduits.

La figure 3 montre la fonction `sol` pour une requête basique avec un filtre. Le filtre est posté avec les contraintes de triplets et élague l'arbre de recherche dès le début. Dans certains cas des propagateurs spécifiques peuvent être utilisés, par exemple pour les opérateurs arithmétiques ou de comparaison. Dans tous les cas, nous pouvons nous rabattre sur un évaluateur d'expressions SPARQL existant pour propager la condition par *forward checking*, c'est à dire lorsque toutes les variables sauf une sont affectées.

Les filtres sur les requêtes composées ne peuvent toutefois être vérifiés qu'après chaque solution de la sous-requête comme le montre la figure 3b. Notez que la condition c n'est pas postée puisqu'il peut y avoir des variables non liées qui ont besoin d'être traitées selon la spécification SPARQL.

Les concaténations sont calculées de façon séquentielle, comme indiqué dans la figure 4a. L'opérateur `OPTIONAL` est similaire à la concaténation et est représenté dans la figure 4b. D'abord $\text{sol}(Q_1)$ est calculé. Avant d'exécuter la seconde sous-requête Q_2, un point de choix est introduit. La branche de gauche calcule $\text{sol}(Q_2)$, fournissant donc les solutions de Q_1 . Q_2. Si elle réussit, la branche de droite est élaguée. Sinon la branche de droite est vide et donc $\text{sol}(Q_1)$ est retourné comme une solution. Notez que cela ne fonctionne qu'avec une recherche en profondeur d'abord explorant la branche de gauche en premier. Enfin pour l'opérateur `UNION`, les deux sous-requêtes sont résolues dans deux branches distinctes, comme le montre la figure 4c.

Il est clair que cette sémantique opérationnelle de requêtes SPARQL calcule l'ensemble de solutions définie par la modélisation déclarative.

5 Castor : un Solveur Léger pour le Web Sémantique

Nous présentons maintenant Castor, un solveur léger, conçu pour résoudre des requêtes SPARQL. Une requête n'implique pas beaucoup de variables et de contraintes. Le principal défi cohére à traiter les domaines immenses associés aux variables. Les solveurs CP existants ne s'adaptent pas bien à ce contexte comme indiqué dans la section expérimentale. L'idée clé de Castor est d'éviter de maintenir et de restaurer des structures de données qui sont proportionnelles aux tailles des domaines. D'une part, nous n'utilisons pas de techniques de propagation avancées qui ont besoin de ce genre de structures coûteuses. D'autre part, la restauration des domaines est une opération peu coûteuse qui nous permet d'explorer de vastes arbres de recherche assez vite pour compenser la perte de propagation.

Dans cette section, nous expliquons les trois principales

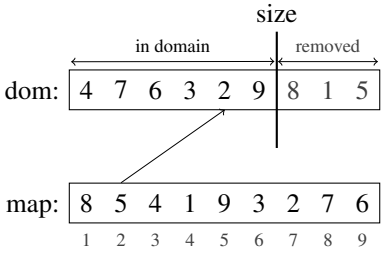


FIGURE 5 – Exemple de représentation du domaine $\{2,3,4,6,7,9\}$, où `size` $= 6$ et où le domaine initial est $\{1,\ldots,9\}$. Les `size` premières valeurs de `dom` appartiennent au domaine ; les valeurs suivantes sont celles qui ont été retirées. Le tableau `map` fait le lien entre les valeurs et leur position dans `dom`. Par exemple, la valeur 2 a l'indice 5 dans `dom`. Dans cette représentation, uniquement la taille doit être restaurée lors d'un retour en arrière.

composantes du solveur : les variables et la représentation de leurs domaines, les contraintes et leur propagateurs, et les techniques de recherche utilisées pour explorer l'arbre.

5.1 Variables et Représentation des Domaines

Les variables dans Castor sont des nombres entiers prenant les valeurs de 1 jusqu'au nombre de valeurs dans la base de données. L'ordre des valeurs est cohérent avec l'opérateur de comparaison de SPARQL. Deux types de représentation sont considérés pour les domaines. La représentation *discrète* maintient chaque valeur du domaine, mais ignore l'ordre des valeurs. La représentation *bornée* ne maintient que la valeur minimale et maximale. Nous proposons une approche duale, tirant parti des points forts des deux représentations.

Représentation discrète. Nous représentons un domaine fini $D(x)$ d'une variable x par sa taille `size` et deux tableaux `dom` et `map`. Les `size` premières valeurs de `dom` sont dans le domaine de la variable, les autres ont été retirées (voir figure 5). Le tableau `map` fait le lien entre les valeurs et leur position dans le tableau `dom`. De telles structures sont aussi utilisées dans le code pour calculer des isomorphismes de sous-graphes décrit dans [20].

Les invariants suivants sont vérifiés.

- Les tableaux `dom` et `map` sont cohérents, c.à.d. $\texttt{map}[v] = i \Leftrightarrow \texttt{dom}[i] = v$.
- Le domaine $D(x)$ est l'ensemble des `size` premières valeurs de `dom`, c.à.d. $D(x) = \{\, \texttt{dom}[i] \mid i \in \{1,\ldots,\texttt{size}\}\,\}$.
- Toute réduction du domaine ne modifie pas les valeurs retirées précédemment (c.à.d., les valeurs de l'indice $\texttt{size}+1$ jusqu'à la fin du tableau `dom`).

Le dernier invariant nous permet de ne restaurer que `size` quand on revient en arrière. En effet, le partitionnement entre les valeurs retirées et les valeurs encore dans le domaine sera identique. L'ordre des valeurs avant `size` peut avoir changé. Le dernier invariant est respecté lors d'une recherche en profondeur d'abord, car nous retirons des valeurs le long d'une branche avant de revenir en arrière.

Les opérations de base sur le domaine ont toutes une complexité en temps constant. Vérifier si une valeur est dans un domaine est réalisé par la propriété $v \in D(x) \Leftrightarrow \texttt{map}[v] \leq \texttt{size}$. Pour supprimer une valeur, nous l'échangeons avec la dernière valeur dans le domaine et nous décrémentons `size`. Par exemple, pour supprimer la valeur 3 dans la figure 5, nous permutons 3 et 9 dans `dom`, mettons à jour `map` en conséquence et diminuons `size` de un.

Pour restreindre le domaine à un ensemble de valeurs, nous *marquons* chacune des valeurs à conserver, c.à.d. nous échangeons la valeur avec la valeur non-marquée la plus à gauche dans `dom` et incrémentons le nombre de valeurs marquées. Nous fixons ensuite la taille du domaine au nombre de valeurs marquées. L'opération complète a une complexité en temps linéaire au nombre de valeurs conservées.

Représentation bornée. Le domaine est représenté par ses bornes, c'est à dire ses valeurs minimales et maximales. Contrairement à la représentation discrète, cette représentation est une approximation du domaine exact. Nous supposons que toutes les valeurs entre les bornes sont présentes dans le domaine.

Dans cette représentation, nous ne pouvons pas supprimer une valeur au milieu du domaine, car nous ne pouvons pas représenter un trou à l'intérieur des limites. Cependant, l'augmentation de la borne inférieure ou la diminution de la borne supérieure se fait en temps constant.

La structure de données pour cette représentation étant de petite taille (seulement deux nombres), nous copions la structure entière à chaque point de choix. Restaurer le domaine cohére à restaurer les deux bornes.

Approche Duale. Les propagateurs réalisant du *forward checking* ou de la cohérence de domaine retirent des valeurs du domaine et ont donc besoin d'une représentation discrète. Cependant, les propagateurs réalisant de la cohérence aux bornes ne mettent à jour que les bornes des domaines. Pour que ceux-ci soient efficaces, nous avons besoin d'une représentation bornée. C'est pourquoi Castor crée deux variables x_D et x_B (resp. avec une représentation discrète et bornée) pour chaque variable SPARQL x. Les contraintes sont postées en utilisant seulement l'une des deux variables, en fonction de la représentation la plus efficace pour le propagateur associé. En particulier, les contraintes monotones sont postées sur des variables bornées, tandis que les contraintes de triplets sont postées sur des variables discrètes.

Une contrainte supplémentaire $x_D = x_B$ assure le lien entre les deux vues. Assurer la cohérence de domaine pour

cette contrainte est trop coûteux, car cela équivaut à effectuer toutes les opérations sur les bornes sur la représentation discrète. Au lieu de cela, le propagateur de Castor fait du *forward checking*, c'est à dire qu'une fois une variable affectée, l'autre sera affectée à la même valeur. Comme optimisation, lors de la restriction d'un domaine à son intersection avec un ensemble S, nous filtrons les valeurs de S qui sont en dehors des bornes et mettons à jour les bornes de x_B. Cette optimisation ne change pas la complexité car S doit être entièrement parcouru de toute façon.

5.2 Contraintes et propagateurs

Il existe deux types de contraintes dans les requêtes SPARQL : les motifs de triplets et les filtres. Les filtres sur des requêtes composées ne sont vérifiés qu'après affectation de toutes leurs variables. Les filtres sur les requêtes basiques et les triplets sont postés et exploités pendant la recherche. Comme pour les domaines, l'objectif est de minimiser les structures devant être restaurées lors d'un retour en arrière. Dans la version actuelle de Castor, les propagateurs n'utilisent pas ce genre de structures.

Une contrainte dans Castor est un objet qui implémente deux méthodes : `propagate` et `restore`. Quand la contrainte est créée, elle s'enregistre aux événements des variables. La méthode `propagate` est appelée lorsque l'un de ces événements se produit. La méthode `restore` est appelée lorsque la recherche fait marche arrière. Actuellement, chaque variable a quatre événements : `bind`, qui se produit lorsque le domaine devient un singleton, `change`, qui se produit lorsque le domaine a changé, `updateMin` et `updateMax`, qui se produisent lorsque la borne inférieure (resp. supérieure) a changé. Pour savoir quelles valeurs ont été retirées d'une variable depuis la dernière exécution du propagateur, on enregistre (localement à la contrainte) la taille du domaine à la fin de la méthode `propagate`. Les valeurs retirées se situent entre l'ancienne et la nouvelle taille dans le tableau `dom` au prochain appel de la méthode. La méthode `restore` est utilisée pour réinitialiser les tailles enregistrées après un retour en arrière. Les propagateurs sont appelés jusqu'à ce que le point fixe soit atteint.

Motifs de triplets. Un motif de triplet est une contrainte table. Elle réagit à l'événement `bind` des variables. Quand une variable est liée, nous cherchons tous les triplets cohérents de la base de données et nous restreignons les domaines des variables non liées restantes. Les triplets sont stockés sur disque en utilisant le schéma introduit par RDF-3x [14], permettant une recherche efficace par index.

Filtres. La vérification des filtres sur les requêtes composées est effectuée par un évaluateur d'expression conforme aux spécifications SPARQL. L'évaluateur considère toutes les variables avec une taille de domaine supérieure à 1 comme non liées. Les filtres sur les requêtes basiques sont postés avec les motifs de triplets. Le propagateur fait du *forward checking* : dès que toutes les variables sauf une sont affectées, nous enlevons du domaine de la variable non affectée les valeurs qui rendent l'expression fausse.

Certains filtres peuvent être propagés plus efficacement avec des algorithmes spécialisés. Le propagateur pour $x \neq y$ attend qu'une valeur soit attribuée à l'une des deux variables et la supprime du domaine de l'autre variable. Il n'est pas besoin d'itérer sur toutes les valeurs du domaine. La contrainte $x = y$ assure la cohérence d'arc en retirant de $D(y)$ les valeurs qui ont été retirées de $D(x)$ et vice-versa, en réagissant à l'événement `change`. De même, le propagateur pour $x < y$ assure la cohérence aux bornes.

5.3 Recherche

L'arbre de recherche est défini en utilisant une stratégie d'étiquetage. À chaque nœud, une variable est choisie et un nœud enfant est créé pour chacune des valeurs de son domaine. L'heuristique standard du plus petit domaine est utilisée pour choisir la variable. L'ordre des valeurs est défini par leur ordre actuel dans le tableau `dom`.

L'arbre de recherche est exploré en profondeur d'abord afin de restaurer efficacement les domaines (section 5.1) et de contrôler efficacement l'incohérence de sous-requêtes optionnelles (section 4).

Pour poster des contraintes lors de la recherche, nous introduisons des *sous-arbres*. Un sous-arbre contient un ensemble de contraintes et un ensemble de variables à étiqueter. Il parcourt toutes les affectations des variables satisfaisant les contraintes. À chaque affectation, Castor peut créer un nouveau sous-arbre ou afficher la solution, en fonction de la requête. Quand un sous-arbre a été complètement exploré, les domaines des variables sont restaurés à leur état lorsque le sous-arbre a été créé et les contraintes sont enlevées, et la recherche continue dans le sous-arbre précédent.

6 Résultats expérimentaux

Pour évaluer la faisabilité et les performances de notre approche, nous avons exécuté des requêtes du SPARQL Performance Benchmark (SP^{2}Bench) [17]. Le SP^{2}Bench cohére en un générateur déterministe de base de données RDF de taille configurable, et 12 requêtes représentatives. Les bases de données représentent les relations entre des articles académiques fictifs et leurs auteurs, selon le modèle de publications académiques dans la base de données DBLP. Le benchmark comporte à la fois des requêtes basiques et composées, mais ne fait usage que de simples filtres de comparaison. Nous avons retiré des requêtes les modificateurs de solution non pris en charge comme `DISTINCT` et `ORDER BY`. Nous nous concentrons sur les requêtes identifiées comme difficiles par les auteurs du

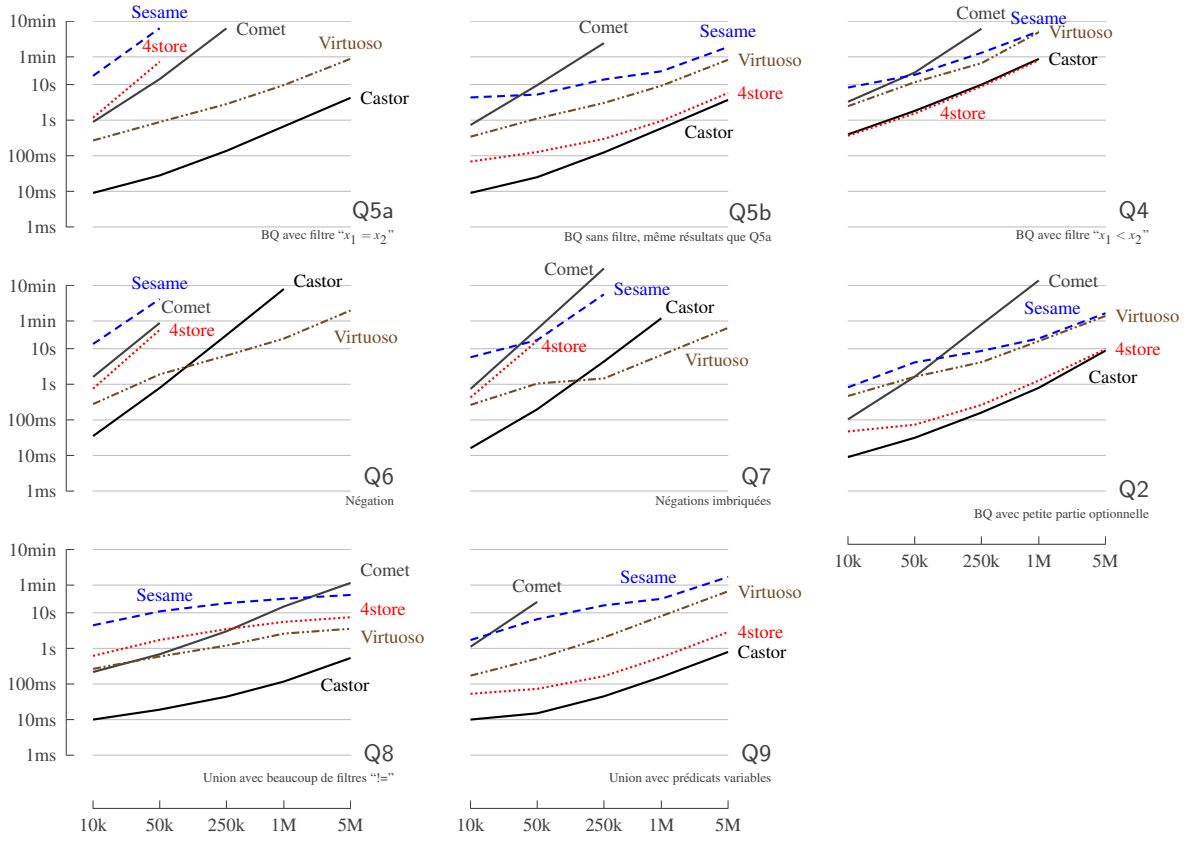


FIGURE 6 – Comparaison sur 8 requêtes. Les abscisses représentent les tailles des bases en nombre de triplets. Les ordonnées donnent les temps d'exécutions des requêtes. Les deux axes ont une échelle logarithmique.

SP^{2}Bench (Q4, Q5, Q6 et Q7) ainsi qu'une requête plus simple (Q2) et deux requêtes impliquant l'opérateur `UNION` (Q8 et Q9). Nous considérons donc 8 requêtes puisqu'il y a deux variantes de Q5.

Nous comparons les performances des moteurs SPARQL de l'état de l'art 4store [9], Sesame [4] et Virtuoso [7], avec le solveur Castor décrit en section 5 et une implémentation directe de la sémantique opérationnelle dans Comet [6]. L'implémentation Comet charge toute la base de données en mémoire. Elle utilise la contrainte table du système pour les motifs de triplets et les expressions du système pour les filtres. Sesame a été exécuté avec le store natif sur disque et trois index (spoc, posc, ospc).

Nous avons généré 6 bases de données de 10k, 25k, 250k, 1M et 5M triplets. Nous avons effectué trois mesures à froid de chaque requête sur toutes les bases de données générées, c.à.d. que les moteurs ont été redémarrés et la cache du système effacé entre deux exécutions. Ces paramètres correspondent à ceux utilisés par les auteurs du SP^{2}Bench. Toutes les expériences ont été effectuées sur un ordinateur Intel Pentium 4 3.2 GHz sous ArchLinux 64bits avec un noyau 3.2.6, 3 Go de RAM DDR-400 et un disque Samsung SP0411C SATA/150 de 40 Go avec un système de fichiers ext4. Nous reprenons le temps écoulé à résoudre les requêtes, sans compter le temps nécessaire pour charger les bases de données. Nous avons vérifié que tous les moteurs trouvent le même ensemble de solutions.

La figure 6 montre le temps d'exécution des requêtes considérées. Notez que les deux axes ont des échelles logarithmiques. Nous discutons maintenant des résultats pour chaque requête.

Requête simple. La requête Q2 est de la forme BQ_1 `OPTIONAL` BQ_2. BQ_1 est une requête basique avec 9 variables et 9 motifs de triplets. La partie optionnelle BQ_2 a un triplet unique avec une seule variable n'apparaissant dans BQ_1. L'exécution de la sous-requête BQ_2 peut donc être faite par un accès à la base de données. 4store et Castor sont de performance égale, surpassant Sesame et Virtuoso. Comet souffre des structures de données lourdes.

Filtres. Les requêtes Q4 et Q5a sont similaires. Les deux sont des requêtes basiques avec un filtre. La requête Q4 a 7 variables, 8 motifs de triplets et un filtre $x_1 < x_2$ sur deux variables x_1 et x_2. La requête Q5a a 6 variables, 6 motifs de triplets et un filtre $x_1 = x_2$. Les moteurs CP sont capables

de dépasser l'état de l'art sur Q5a grâce à leur propagation efficace de la contrainte d'égalité. Nous soupçonnons que cette contrainte soit traitée en post-processing dans Sesame et 4store. Virtuoso obtient de meilleures performances en optimisant la requête, mais cela ne préserve pas correctement la sémantique. La requête Q4 est plus difficile car elle a beaucoup plus de solutions que Q5a ($2.65 \cdot 10^6$ contre $1.01 \cdot 10^5$ pour la base de données avec 1M triplets). Ainsi, le filtrage est moins contraignant. Castor est toujours compétitif avec l'état de l'art.

Les deux variantes de Q5, Q5a et Q5b, calculent exactement le même ensemble de solutions. La dernière encode la contrainte d'égalité dans ses 5 motifs de triplets en utilisant 4 variables. Sans surprise, les moteurs CP fonctionnent de manière semblable sur les deux requêtes comme ils exploitent les filtres dès le début pendant la recherche. Sesame et 4store gèrent bien mieux la requête sans filtre que Q5a. Ceci montre la pertinence de notre approche, particulièrement compte tenu du fait que les filtres sont présents dans environ la moitié des requêtes dans le monde réel [2].

Négations. Une négation en SPARQL est une requête composée qui a la forme $(Q_1 \texttt{ OPTIONAL } Q_2)$ `FILTER` $(!\text{bound}(x))$, où x est une variable n'apparaissant que dans Q_2. Le filtre élimine toutes les solutions affectant une valeur à x, c'est à dire que nous ne gardons que les solutions de Q_1 qui ne peuvent pas être étendues à des solutions de $Q_1 \,.\, Q_2$. La requête Q6 est une négation avec des filtres additionnels à l'intérieur de Q_2. La requête Q7 n'a pas de filtres supplémentaires, mais Q_2 est elle-même une négation imbriquée. Sur les deux requêtes, Castor surpasse Sesame et 4store, mais n'est pas meilleur que Virtuoso.

Unions. Les requêtes composées Q8 et Q9 utilisent l'opérateur `UNION`. La première ajoute des filtres d'inégalité dans ses deux sous-requêtes. Les sous-requêtes de la dernière ne contiennent que deux motifs de triplets chacun. Pourtant, Q9 génère de nombreuses solutions. Comet est incapable d'aller au delà de 50k triplets en raison de ses structures lourdes. Castor surpasse toutefois l'état de l'art. Dans la requête Q8, les deux sous-requêtes alternatives ont certains motifs de triplets dupliqués. L'exploitation de cette propriété peut expliquer l'aspect plat des courbes de l'état de l'art par rapport à Castor.

Conclusion. La table 1 montre la vitesse relative de Castor par rapport à 4 store et Virtuoso. Le but de Castor est d'utiliser la CP pour résoudre des requêtes très contraintes, c.à.d. des requêtes où les filtres éliminent beaucoup de solutions. Ces requêtes (par exemple Q5a) sont traitées beaucoup plus efficacement par Castor. Sur les requêtes reposant plus sur l'accès base de données (comme Q2 et Q9), la CP reste compétitive.

7 Discussion

Nous avons proposé une modélisation déclarative et une sémantique opérationnelle pour résoudre les requêtes SPARQL en utilisant la programmation par contraintes. Nous avons introduit un solveur léger spécialisé implémentant cette sémantique. Nous avons montré que cette approche surpasse l'état de l'art sur des requêtes contraintes, et est compétitive sur la plupart des autres requêtes.

Travaux apparentés. Mamoulis et Stergiou ont utilisé des CSPs pour résoudre des requêtes XPath complexes sur des documents XML [12]. Les documents XML sont des graphes, tout comme les données RDF, mais avec une structure d'arbre sous-jacente. Cette structure est utilisée par les auteurs pour concevoir des propagateurs spécifiques. Cependant, ils ne peuvent pas être utilisés pour les requêtes SPARQL. Mouhoub et Feng ont appliqué la programmation par contraintes à la résolution de requêtes combinatoires dans des bases de données relationnelles [13]. Ces requêtes impliquent de joindre plusieurs tables soumises à des contraintes arithmétiques complexes. Le problème est similaire à SPARQL. Toutefois, les auteurs ne traitent pas les bases volumineuses. Leurs expériences sont limitées aux tables avec 800 lignes. Cette taille n'est pas réaliste pour des données RDF. D'autres travaux visent à étendre le langage de requête SQL standard pour soutenir des expressions de satisfaction de contraintes explicites [11, 19]. Cela permet de résoudre des CSPs dans des bases de données relationnelles.

Travaux futures. Castor est un jeune prototype qui a vocation à être amélioré et étendu. Par exemple, l'heuristique de sélection de variables est le MinDom de base. Une autre solution pourrait être de choisir en premier les variables centrales des requêtes en forme d'étoile. En outre, aucune recherche n'a encore été faite pour trouver un ordonnancement optimal des propagateurs : ils sont simplement appelés successivement jusqu'au point fixe. Pour atteindre le point fixe plus rapidement, il serait préférable d'appeler d'abord les propagateurs élaguant le plus. Les estimations de sélectivité, un outil utilisé dans les bases de données relationnelles [21], pourraient peut-être être utilisés pour ordonner les propagateurs. Cela soulève la question plus générale de savoir si et comment les techniques d'optimisations utilisées dans les bases de données relationnelles peuvent être combinées avec l'approche CP.

Remerciements. Le premier auteur est financé comme assistant de recherche par le FNRS belge (fonds national de la recherche scientifique). Cette recherche est aussi financée par le programme de pôles d'attraction interuniversitaire (état belge, police scientifique belge) et le projet FRFC 2.4504.10 du FNRS belge.

TABLE 1 – Speedup de Castor par rapport à 4store (à gauche) et Virtusoso (à droite). La lettre 'C' (resp. 'V') signifie que seul Castor (resp. Virtuoso) était capable de résoudre l'instance dans la limite de temps.

	10k	50k	250k	1M	5M
Q2	5.39	2.34	1.64	1.59	1.10
Q4	0.92	0.85	0.89	0.92	—
Q5a	133.39	1574.62	C	C	C
Q5b	7.87	5.11	2.40	1.61	1.58
Q6	21.45	42.86	C	C	—
Q7	26.87	84.14	C	C	—
Q8	61.67	91.40	78.80	47.49	13.99
Q9	5.30	4.91	3.69	3.56	3.64

	10k	50k	250k	1M	5M
Q2	53.90	52.02	26.08	20.47	9.42
Q4	6.20	6.41	4.01	5.64	—
Q5a	31.01	31.71	20.66	14.45	12.48
Q5b	39.12	44.63	24.50	15.68	13.37
Q6	7.94	2.40	0.26	0.04	V
Q7	16.92	5.30	0.34	0.09	V
Q8	26.57	31.36	27.36	22.18	6.65
Q9	17.13	34.86	45.09	51.23	50.62

Références

[1] Renzo Angles and Claudio Gutierrez. The expressive power of SPARQL. In *The Semantic Web – ISWC 2008*, volume 5318 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2008.

[2] M. Arias, Javier D. Fernández, Miguel A. Martínez-Prieto, and Pablo de la Fuente. An empirical study of real-world SPARQL queries. In *1st International Workshop on Usage Analysis and the Web of Data (USEWOD 2011), in conjunction with WWW 2011*, 2011.

[3] Jean-François Baget. RDF entailment as a graph homomorphism. In *The Semantic Web – ISWC 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 2005.

[4] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame : A generic architecture for storing and querying RDF and RDF Schema. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC '02, pages 54–68. Springer, 2002.

[5] Michael J. Cafarella, Alon Halevy, and Jayant Madhavan. Structured data on the web. *Commun. ACM*, 54 :72–79, February 2011.

[6] Dynamic Decision Technologies Inc. *Comet*, 2010.

[7] Orri Erling and Ivan Mikhailov. RDF support in the Virtuoso DBMS. In *Networked Knowledge – Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 7–24. Springer, 2009.

[8] Gecode Team. Gecode : Generic constraint development environment, 2006.

[9] Steve Harris, Nick Lamb, and Nigel Shadbolt. 4store : The design and implementation of a clustered RDF store. In *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2009), at ISWC 2009*, 2009.

[10] G. Klyne, J. J. Carroll, and B. McBride. Resource description framework (RDF) : Concepts and abstract syntax, 2004.

[11] Robin Lohfert, James Lu, and Dongfang Zhao. Solving SQL constraints by incremental translation to SAT. In *New Frontiers in Applied Artificial Intelligence*, volume 5027 of *Lecture Notes in Computer Science*, pages 669–676. Springer, 2008.

[12] Nikos Mamoulis and Kostas Stergiou. Constraint satisfaction in semi-structured data graphs. In *Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *Lecture Notes in Computer Science*, pages 393–407. Springer, 2004.

[13] Malek Mouhoub and Chang Feng. CSP techniques for solving combinatorial queries within relational databases. In *Intelligent Systems for Knowledge Management*, volume 252 of *Studies in Computational Intelligence*, pages 131–151. Springer, 2009.

[14] Thomas Neumann and Gerhard Weikum. RDF-3X : a RISC-style engine for RDF. *Proc. VLDB Endow.*, 1 :647–659, August 2008.

[15] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34 :16 :1–16 :45, September 2009.

[16] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF, January 2008.

[17] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP^2Bench : A SPARQL performance benchmark. In *Proc. IEEE 25th Int. Conf. Data Engineering ICDE '09*, pages 222–233, 2009.

[18] Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL query optimization. In *Proceedings of the 13th International Conference on Database Theory*, ICDT '10, pages 4–33. ACM, 2010.

[19] Sebastien Siva and Lesi Wang. A SQL database system for solving constraints. In *Proceeding of the 2nd PhD workshop on Information and knowledge management*, PIKM '08, pages 1–8. ACM, 2008.

[20] Christine Solnon. Alldifferent-based filtering for subgraph isomorphism. *Artificial Intelligence*, 174(12–13) :850–864, 2010.

[21] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. SPARQL basic graph pattern optimization using selectivity estimation. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 595–604, New York, NY, USA, 2008. ACM.

Propagation des contraintes tables souples Etude préliminaire

Christophe Lecoutre **Nicolas Paris** **Olivier Roussel** **Sébastien Tabary**

CRIL - CNRS UMR 8188,
Université Lille Nord de France, Artois,
rue de l'universite, 62307 Lens cedex, France
{lecoutre,paris,roussel,tabary}@cril.fr

Résumé

Durant ces dix dernières années, de nombreuses études ont étés réalisées pour le cadre WCSP (Weighted Constraint Satisfaction Problem). En particulier, ont été proposées de nombreuses techniques de filtrage basées sur le concept de cohérence locale souple telle que la cohérence de nœud, et surtout la cohérence d'arc souple. Toutefois, la plupart de ces algorithmes ont étés introduits pour le cas des contraintes binaires, et la plupart des expérimentations ont étés menées sur des réseaux de contraintes comportant uniquement des contraintes binaires et/ou ternaires. Dans cet article, nous nous intéressons aux contraintes tables souples de grande arité. Nous proposons un premier algorithme pour filtrer ces contraintes et nous l'intégrons à PFC-MRDAC. Bien que nous soyons encore dans la phase préliminaire de nos travaux, nous montrons que cet algorithme peut être utile pour résoudre efficacement un nouveau jeu d'instances de mots-croisés.

Abstract

WCSP is a framework that has attracted a lot of attention during the last decade. In particular, there have been many developments of filtering approaches based on the concept of soft local consistencies such as node consistency (NC), arc consistency (AC), full directional arc consistency (FDAC), existential directional arc consistency (EDAC), virtual arc consistency (VAC) and optimal soft arc consistency (OSAC). Almost all algorithms related to these properties have been introduced for binary weighted constraint networks, and most of the conducted experiments typically include constraint networks involving only binary and ternary constraints. In this paper, we focus on extensional soft constraints of large arity. We propose an algorithm to filter such constraints and embed it in PFC-MRDAC. Despite still being preliminary at this stage, we show it can be used to solve efficiently a new benchmark of Crossword puzzles.

1 Introduction

Le problème de satisfaction de contraintes pondéré (WCSP) consiste, pour un réseau constitué de variables et de contraintes souples, à trouver une affectation de valeur à chaque variable qui soit optimale, c'est à dire une assignation complète de coût minimal parmi toutes les assignations complètes possibles. Il s'agit d'un problème NP-difficile. Un certain nombre de travaux ont étés menés pour adapter au cadre WCSP des propriétés (et algorithmes efficaces) définies pour le cadre CSP, généralement dans le but de filtrer l'espace de recherche, comme par exemple la cohérence de nœud (NC) ou la cohérence d'arc (AC) [11, 14]. Un certain nombre d'algorithmes de plus en plus sophistiqués ont été proposés au cours des années pour approcher la cohérence d'arc souple idéale : FDAC, EDAC, VAC et OSAC (voir [6]).

Les algorithmes évoqués ci-dessus utilisent des opérations de transfert de coûts, ce qui les rend particulièrement efficaces pour résoudre des instances de problèmes binaires et/ou ternaires. Pour des contraintes de plus grande arité, un problème d'ordre combinatoire se présente. Une première solution à ce problème est de retarder la propagation (des coûts) en attendant qu'un nombre suffisant de variables soient affectées, mais cela réduit considérablement la capacité de filtrage des algorithmes en début de recherche. Une seconde solution est de concevoir des adaptations des algorithmes de cohérence d'arc souple pour certaines familles de contraintes (globales) souples. C'est l'approche proposée dans [17] où le concept de contraintes

saines pour la projection est introduit. Enfin, une troisième solution [9] est la décomposition de fonctions de coûts en fonction de coûts d'arité plus faible. Toutefois, toutes les fonctions de coût ne sont pas décomposables. Dans cet article, nous proposons une première approche pour les contraintes tables souples (i.e. les contraintes souples définies en extension) en exploitant le principe de la réduction tabulaire simple (STR) [19]. À ce stade, notre approche ne permet pas les transferts de coût et se trouve donc intégrée à l'algorithme classique PFC-MRDAC [12].

2 Préliminaires

Un *réseau de contraintes* (CN) P est un couple $(\mathscr{X}, \mathscr{C})$ où $\mathscr{X}$ est un ensemble fini de n variables et $\mathscr{C}$ est un ensemble fini de e contraintes. Chaque variable x a un domaine associé noté $dom(x)$, qui contient l'ensemble fini des valeurs pouvant être assignées à x ; le domaine initial de x est noté $dom^{init}(x)$. d représente la taille du plus grand domaine. Chaque contrainte c_S porte sur un ensemble ordonné $S \subseteq \mathscr{X}$ de variables appelé *portée* de c_S, et définie par une relation contenant l'ensemble des tuples autorisés pour les variables de S. L'arité d'une contrainte est le nombre de variables dans sa portée. Une contrainte *unaire* (resp., *binaire*) porte sur 1 (resp., 2) variable(s), et une contrainte *non-binaire* porte sur plus de deux variables. Une *instanciation* I d'un ensemble $X = \{x_1, \ldots, x_p\}$ de variables est un ensemble $\{(x_1, a_1), \ldots, (x_p, a_p)\}$ tel que $\forall i \in 1..p, a_i \in dom^{init}(x_i)$; chaque a_i est noté $I[x_i]$. I est *valide* sur P ssi $\forall (x, a) \in I, a \in dom(x)$. Une solution de P est une instanciation complète de P (i.e., l'assignation d'une valeur à chaque variable) qui satisfait toutes les contraintes. Pour plus d'information sur les réseaux de contraintes, voir [8, 15].

Un *réseau de contraintes pondéré* (WCN) P est un triplet $(\mathscr{X}, \mathscr{C}, k)$ où $\mathscr{X}$ est un ensemble fini de n variables, comme pour un CN, $\mathscr{C}$ est un ensemble fini de e contraintes souples, et k est soit un entier naturel strictement positif soit $+\infty$. Chaque contrainte souple $c_S \in \mathscr{C}$ porte sur un ensemble ordonné S de variables (sa portée) et est définie par une fonction de coût de $l(S)$ vers $\{0, \ldots, k\}$, où $l(S)$ est le produit cartésien des domaines des variables présentes dans S ; pour toute instanciation $I \in l(S)$, on notera le coût de I dans c_S par $c_S(I)$. Pour simplifier, pour toute contrainte (souple) c_S, un couple (x, a) avec $x \in S$ et $a \in dom(x)$ est appelé une *valeur* de c_S. Une instanciation de coût k (noté aussi $\top$) est interdite. Autrement, elle est autorisée avec le coût correspondant (0, noté aussi $\bot$, est complètement satisfaisant). Les coûts sont combinés par l'opérateur binaire $\oplus$ défini par :

$$\forall a, b \in \{0, \ldots, k\}, a \oplus b = min(k, a + b)$$

L'objectif du problème de satisfaction de contraintes pondéré (WCSP) est, pour un WCN donné, de trouver une instanciation complète de coût minimal. Pour plus d'information sur les contraintes pondérées (valuées), voir [2, 18].

Différentes variantes de la cohérence d'arc souple pour le cadre WCSP ont étés proposées durant ces dix dernières années. Il s'agit de AC* [11, 14], la cohérence d'arc directionnelle complète (FDAC) [4], la cohérence d'arc directionnelle existentielle (EDAC) [7], la cohérence d'arc virtuelle (VAC) [5] et la cohérence d'arc souple optimale (OSAC) [6]. Tous les algorithmes proposés pour atteindre ces différents niveaux de cohérence utilisent des opérations de transfert de coûts (ou transformations préservant l'équivalence) telles que la projection unaire, la projection et l'extension. Une autre approche classique consiste à ne pas effectuer de transferts de coûts. Il s'agit de l'algorithme PFC-MRDAC [10, 13, 12] qui est un algorithme de séparation et évaluation (branch and bound) permettant de calculer des bornes inférieures à chaque nœud de l'arbre de recherche. À chaque nœud, on calcule un minorant (lb), sous estimation du coût de n'importe quelle solution pouvant être atteinte à partir du nœud courant. On considère une partition [1] *part* de l'ensemble des contraintes qui à chaque variable x associe un élément noté $part(x)$ de cette partition (i.e., chaque contrainte est associée à exactement une variable). Pour une valeur (x, a), il est possible de calculer un minorant $lb(x, a)$ comme suit : $lb(x, a) =$

$$distance + \tag{1}$$

$$\sum_{c_S \in part^{nc}(x)} minCosts(c_S, x, a) + \tag{2}$$

$$\sum_{y \neq x} \min_{b \in dom(y)} \sum_{c'_S \in part^{nc}(y)} minCosts(c'_S, y, b) \tag{3}$$

où :

- *distance* représente le coût des contraintes couvertes, i.e. la somme du coût des contraintes dont la portée ne comporte que des variables instanciées (explicitement par l'algorithme de recherche arborescente) ;
- $part^{nc}(x)$ représente $part(x)$ sans les contraintes couvertes (pour ne pas les compter plus d'une fois) ;
- $minCosts(c_S, x, a)$ représente le coût minimal dans c_S de toute instanciation valide comportant (x, a) ; plus formellement $minCosts(c_S, x, a) = min\{c_S(I) \mid I \in l(S) \wedge I[x] = a\}$.

1. On suppose que chaque variable est impliquée dans au moins une contrainte.

La valeur de $lb(x, a)$ est calculée en prenant en compte le coût des contraintes couvertes (équation 1), celui des contraintes (non couvertes) associées à x (équation 2), et celui des contraintes (non couvertes) associées à d'autres variables que x (équation 3). Si $lb(x, a)$ est supérieur ou égal à la borne supérieure (ub) qui représente le coût de la meilleure solution trouvée, alors il est possible de supprimer (x, a).

3 Algorithme

Dans le cadre CSP, une contrainte table (positive) c_S est une contrainte définie en extension par la liste de ses tuples autorisés ; nous notons celle-ci $table[c_S]$ et utilisons comme indices $1..t$ où t désigne le nombre de tuples. L'un des algorithmes de filtrage les plus efficaces pour les contraintes table (pour le cadre CSP) est appelé STR [19, 16]. STR permet de maintenir en permanence la liste des supports de chaque contrainte table ; cette liste est appelée la table courante. Pour une contrainte table donnée, cette liste est gérée à l'aide des structures suivantes :

- $position[c_S]$ est un tableau de taille t qui fournit un accès indirect aux tuples de $table[c_S]$. À tout moment, les valeurs dans $position[c_S]$ représentent une permutation de $\{1, 2, \ldots, t\}$. Le $i^{\text{ème}}$ tuple de la contrainte est $table[c_S][position[c_S][i]]$.
- $currentLimit[c_S]$ est la position du dernier tuple (support) courant dans $table[c_S]$. La table courante de c_S contient exactement $currentLimit[c_S]$ tuples. Les valeurs dans $position[c_S]$ aux indices allant de 1 à $currentLimit[c_S]$ sont les positions des tuples courants de c_S.

Une contrainte table souple c_S est une contrainte définie par une liste $table[c_S]$ de tuples accompagnée d'une liste $costs[c_S]$ de leur coût, ainsi que d'un coût par défaut $default[c_S]$ pour tous les tuples non représentés explicitement. L'algorithme PFC-MRDAC nécessite de calculer pour toute contrainte c_S et toute valeur (x, a) de c_S le coût minimal de tout tuple valide τ tel que $\tau[x] = a$. Nous montrons comment effectuer ce calcul en adaptant STR aux contraintes tables souples grâce aux structures suivantes :

- $minCosts[c_S]$ est un tableau (à 2 dimensions) qui donne le coût minimal de tout tuple pour chaque valeur (x, a) de c_S.
- $nbTuples[c_S]$ est un tableau (à 2 dimensions) qui donne le nombre de tuples valides (parmi ceux représentés explicitement dans la table initiale) pour chaque valeur (x, a) de c_S.

WSTR, i.e. STR pour le cadre WCSP, est décrit par l'algorithme 1. Cet algorithme est appelé chaque fois qu'un évènement nécessite de mettre à jour la table courante et donc les coûts minimaux de chaque

Algorithm 1: WSTR(c_S : contrainte souple)

```
1  foreach variable x ∈ S do
2      foreach a ∈ dom(x) do
3          minCosts[c_S][x][a] ← ub
4          nbTuples[c_S][x][a] ← 0
5  i ← 1
6  while i ≤ currentLimit[c_S] do
7      index ← position[c_S][i]
8      τ ← table[c_S][index]
9      γ ← costs[c_S][index]
10     if isValidTuple(c_S, τ) then
11         foreach variable x ∈ S do
12             a ← τ[x]
13             nbTuples[c_S][x][a] + +
14             if γ < minCosts[c_S][x][a] then
15                 minCosts[c_S][x][a] ← γ
16         i ← i + 1
17     else
18         swap(position[c_S], i, currentLimit[c_S])
19         currentLimit[c_S] − −
20 foreach variable x ∈ S do
21     nb ← |Π_{y∈scp(c)|y≠x} dom(y)|
22     foreach a ∈ dom(x) do
23         if nbTuples[c_S][x][a] ≠ nb then
24             if default[c_S] < minCosts[c_S][x][a]
               then
25                 minCosts[c_S][x][a] ← default[c_S]
```

valeur. Tout d'abord, les structures $minCosts[c_S]$ et $nbTuples[c_S]$ sont initialisées aux lignes 1 à 4. Ensuite, chaque tuple τ (ligne 8) de la table courante et son coût γ (ligne 9) sont considérés. Si le tuple est toujours valide (ligne 10, appel à l'algorithme 2) alors on met à jour les structures $minCosts[c_S]$ et $nbTuples[c_S]$ (lignes 11 à 15). Sinon, on élimine le tuple en échangeant la position des tuples aux positions i et $currentLimit[c_S]$ (ligne 18) et en décrémentant ensuite $currentLimit[c_S]$ (ligne 19). Pour finir, il faut prendre en compte les tuples non représentés explicitement (lignes 20 à 25) : si le nombre de tuples valides trouvé pour une valeur

Algorithm 2: isValidTuple(c_S, τ) : booléen

```
1  foreach variable x ∈ S do
2      if τ[x] ∉ dom(x) then
3          return false
4  return true
```

(x, a) dans la table courante est différent du nombre de tuples valides possibles (calculé à la ligne 21) pour cette valeur alors il existe au moins un tuple pour (x, a) avec ce coût et donc le coût par défaut doit être considéré.

La complexité spatiale de WSTR pour une contrainte table souple c_S, avec r désignant l'arité de c_S et t désignant le nombre de tuples explicites de la table (initiale) de c_S, est donnée par la complexité spatiale des structures de données propres à STR, à savoir $O(n + rt)$ [16], et la complexité spatiale des structures additionnelles $minCosts[c_S]$ et $nbTuples[c_S]$ qui est $O(rd)$. On obtient donc globalement $O(e(n + r.max(d, t)))$. La complexité temporelle de l'algorithme 1 est $O(rd)$ pour les lignes 1-4, $O(tr)$ pour les lignes 5-19, et $O(rd)$ pour les lignes 20-25, soit globalement $O(rd + tr)$.

Un certain nombre de précisions sont à apporter sur cet algorithme. Tout d'abord, dans cet article, pour simplifier il est présenté dans le contexte d'une utilisation hors recherche. Ensuite, il est certainement possible d'apporter un certain nombre d'améliorations selon la distribution des coûts (en particulier, lorsque $defaut[c_S] = 0$ ou $defaut[c_S] = k$). Pour finir, l'intérêt de cet algorithme réside dans son efficacité lorsqu'une réduction importante de la table est constatée, et la possibilité de restaurer les tuples en temps constant.

4 Résultats expérimentaux

Bien sûr, nous avons cherché à valider expérimentalement cette première approche que nous proposons pour filtrer les contraintes tables souples. Malheureusement, à notre connaissance aucun jeu d'essai comportant des instances WCSP avec des contraintes tables d'arité importante n'est disponible. Nous avons donc développé une nouvelle série d'instances de mots-croisés, appelée *crossoft*, qui se prêtent naturellement à une expression "souple". Pour cela, nous avons fusionné deux dictionnaires (appelés OGD), l'un comportant des noms communs et l'autre des noms propres. Le coût d'un mot (tuple) a été calculé de la manière suivante :

- mot commun : coût 0
- mot propre : coût r où r est la longueur du mot

À l'aide de ce nouveau dictionnaire (souple), l'objectif est alors de remplir des grilles de mots-croisés vides en minimisant le coût global. Le type de pénalités considérées ici ne correspond pas exactement aux bénéfices associés aux mots tels que décrits sur le site `http://ledefi.pagesperso-orange.fr` mais il s'en rapproche (et a le mérite de la simplicité). Nous avons généré des instances pour les séries de grilles appelées *herald*, *puzzle*, et *vg*.

Notre premier objectif ici est de montrer que l'approche WSTR peut s'avérer efficace pour résoudre des instances assez difficiles (au regard de la difficulté des instances dures). Nous avons aussi voulu nous assurer que retarder la propagation des algorithmes de filtrage AC* et EDAC (c'est-à-dire attendre qu'il reste au plus 3 ou 4 variables non assignées avant de les exécuter) n'était pas une solution viable (ne pas retarder la propagation l'est encore moins par rapport aux instances que nous avons testées, étant données l'arité des contraintes et la taille des domaines des variables). C'est pourquoi, pour montrer le potentiel de WSTR, nous avons comparé les performances des algorithmes suivants :

- WSTR, l'algorithme 1 utilisé au sein de PFC-MRDAC
- maintenir AC*, avec propagation retardée
- maintenir EDAC, avec propagation retardée

L'heuristique de choix de variable que nous avons choisie est *dom/ddeg* [1] (plus stable que *dom/wdeg* [3] dans un but de comparaison), et l'heuristique de choix de valeur consiste à toujours prendre une valeur de coût minimum. Les expérimentations ont été menées grâce à notre solveur AbsCon sur un cluster équipé de processeurs Intel(R) Xeon(TM) CPU 3.00GHz. Le temps limite alloué pour résoudre chaque instance est de 1200 secondes. Résoudre une instance signifie ici trouver une solution optimale et prouver qu'il s'agit de l'optimum.

Le tableau 1 fournit quelques résultats concernant la résolution des instances *crossoft*. Ces instances comportent toutes des contraintes d'arité supérieure ou égale à 5. Pour chaque instance (à noter que r_{max} désigne l'arité maximale), le temps CPU (cpu) total pour la résoudre est donné ainsi que le nombre de nœuds explorés (nœuds). Les résultats obtenus montrent bien que WSTR est plus efficace que maintenir AC* et EDAC (avec propagation retardée) sur des problèmes de grande arité. Il ne faut toutefois pas être surpris par ces résultats, puisque nous savons que AC* et EDAC (sous leurs formes génériques) ne sont pas adaptés à ces types de problèmes.

5 Conclusion

Dans ce papier, nous avons montré que l'algorithme de filtrage WSTR, intégré à l'algorithme classique PFC-MRDAC, est adapté aux instances WCSP comportant des contraintes tables souples de grande arité. Sans doute, de nombreuses améliorations sont possibles, incluant la possibilité de spécialiser l'algorithme selon la distribution des coûts, et aussi la possibilité sous certaines hypothèses d'utiliser les opérations de transferts de coûts.

Instances	n	e	d	r_{max}		AC*	EDAC	WSTR
ogd-05-03	21	10	26	5	cpu	$1,77$	$1,94$	$0,8$
					nœuds	46	46	182
ogd-05-08	21	10	26	5	cpu	> 1200	$2,81$	$0,75$
					nœuds		317	102
ogd-05-09	19	10	26	5	cpu	$0,98$	> 1200	$0,74$
					nœuds	37		63
ogd-puzzle-04	19	10	26	5	cpu	$1,2$	$1,47$	$0,82$
					nœuds	19	19	54
ogd-puzzle-05	21	10	26	5	cpu	> 1200	> 1200	$0,8$
					nœuds			88
ogd-puzzle-11	38	18	26	7	cpu	$20,7$	> 1200	$1,75$
					nœuds	$271K$		553
ogd-vg-4-7	28	11	26	7	cpu	$14,0$	$14,1$	$1,17$
					nœuds	7351	7351	222
ogd-vg-4-8	32	12	26	8	cpu	645	> 1200	$1,48$
					nœuds	$1398K$		513
ogd-vg-5-5	25	10	26	5	cpu	> 1200	> 1200	$0,9$
					nœuds			232

TABLE 1 – Temps CPU et nombre de nœuds visités pour prouver l'optimalité sur quelques instances *crossoft*.

Remerciements

Ce travail bénéficie du soutien du CNRS et d'OSEO dans le cadre du projet ISI Pajero.

Références

[1] C. Bessiere and J. Régin. MAC and combined heuristics : two reasons to forsake FC (and CBJ ?) on hard problems. In *Proceedings of CP'96*, pages 61–75, 1996.

[2] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs : Frameworks, properties, and comparison. *Constraints*, 4(3) :199–240, 1999.

[3] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.

[4] M. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3) :311–342, 2003.

[5] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, and M. Zytnicki. Virtual arc consistency for weighted CSP. In *Proceedings of AAAI'08*, pages 253–258, 2008.

[6] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8) :449–478, 2010.

[7] S. de Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency : Getting closer to full arc consistency in weighted CSPs. In *Proceedings of IJCAI'05*, pages 84–89, 2005.

[8] R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[9] A. Favier, S. de Givry, A. Legarra, and T. Schiex. Pairwise decomposition for combinatorial optimization in graphical models. In *Proceedings of IJCAI'11*, pages 2126–2132, 2011.

[10] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3) :21–70, 1992.

[11] J. Larrosa. Node and arc consistency in weighted CSP. In *Proceedings of AAAI'02*, pages 48–53, 2002.

[12] J. Larrosa and P. Meseguer. Partition-Based lower bound for Max-CSP. *Constraints*, 7 :407–419, 2002.

[13] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107(1) :149–163, 1999.

[14] J. Larrosa and T. Schiex. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, 159(1-2) :1–26, 2004.

[15] C. Lecoutre. *Constraint networks : techniques and algorithms*. ISTE/Wiley, 2009.

[16] C. Lecoutre. STR2 : Optimized simple tabular reduction for table constraint. *Constraints*, 16(4) :341–371, 2011.

[17] J. Lee and K. Leung. Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. In *Proceedings of IJCAI'09*, pages 559–565, 2009.

[18] P. Meseguer, F. Rossi, and T. Schiex. Soft constraints. In *Handbook of Constraint Programming*, chapter 9, pages 281–328. Elsevier, 2006.

[19] J.R. Ullmann. Partition search for non-binary constraint satisfaction. *Information Science*, 177 :3639–3678, 2007.

Propagation de contraintes arithmétiques

Arnaud Malapert et Jean-Charles Régin
I3S CNRS – Université Nice-Sophia Antipolis
{arnaud.malapert,jean-charles.regin}@unice.fr

Résumé

Nous nous intéressons à la résolution de problèmes de grandes tailles contenant principalement des contraintes arithmétiques comme des problèmes de plus court chemin. Nous montrons qu'un simple modèle PPC n'est pas compétitif avec des algorithmes ou contraintes spécialisés. Ce phénomène est causé par le mécanisme de propagation de contraintes qui détermine l'ordre dans lequel les contraintes sont révisées. Fort de cette observation, nous proposons de modifier la propagation pour intégrer certaines idées cruciales, mais simples, des algorithmes spécialisés. Nous montrons l'intérêt de cette idée sur des problèmes de plus court chemin et nous questionnons sa généralisation à travers des expériences sur d'autres problèmes. Nous analysons aussi la dynamique du phénomène de propagation et constatons que le nombre de variables traitées plusieurs fois dans une même phase de propagation est faible.

Abstract

We consider the resolution by constraint programming of large problems, *i.e.* involving millions of constraints, which mainly imply arithmetic constraints, like shortest path problems or other related problems. We show that a simple constraint programming model is not competitive with dedicated algorithms (or dedicated constraints). This mainly comes from the propagation mechanism, *i.e.* the ordering along which the constraints are revised. Thus, we propose a modification of this propagation mechanism integrating the main ideas of the dedicated algorithms. We give some experiments for the shortest path problem and more general problems which confirms the robustness of our approach. Last, we give some results showing that only a few variables are considered more than once during a propagation step.

1 Introduction

Le problème du plus court chemin (PCC) est une composante de problèmes plus généraux comme les problèmes de cheminement avec contraintes de ressources ou des problèmes d'ordonnancement (PERT/CPM). Nous nous intéresserons à la variante qui consiste à trouver les chemins de longueurs (sommes des longueurs de leurs arcs) minimales d'un nœud racine s vers les autres nœuds dans un graphe orienté. Ce problème peut être résolu efficacement par une méthode de marquage (*labeling method*) utilisant une des deux stratégies classiques pour sélectionner la prochaine variable à examiner, celle de Bellman-Ford-Moore [2] et celle de Dijkstra [6]. Avant de résoudre des problèmes plus généraux, nous comparerons d'abord les performances de la programmation par contraintes (PPC) par rapport aux algorithmes spécialisés. Nous montrerons en fait que la conception d'un modèle simple compétitif avec ces algorithmes est loin d'être évidente. Ces piètres performances sont principalement dues au mécanisme de propagation de contraintes dont nous modifierons en conséquence l'initialisation et l'ordre de traitement des variables.. Puis, nous évaluerons l'impact de ces modifications sur les *radio link frequency assignment problems* (RLFAP). Finalement, nous analyserons des résultats concernant le nombre de fois où une variable est traitée pendant une phase de propagation.

2 Méthode de marquage et PPC

La méthode de marquage et la propagation de contraintes ont des ressemblances frappantes pour la résolution du PCC. Soit $G = (X, A)$ un graphe orienté où chaque arc $(u, v) \in A$ a une longueur $l(u, v)$, et soit s un nœud de X. Nous cherchons à calculer les plus courts chemins depuis ce nœud racine s vers tous les autres nœuds $v \in X$.

Méthode de marquage La méthode de marquage est définie de la manière suivante [5]. Pour chaque nœud v, la méthode maintient sa distance $d(v)$, son parent $p(v)$ et son statut $S(v) \in \{unreached, labeled, scanned\}$.

Lors de l'initialisation, ces valeurs sont fixées de la manière suivante pour chaque nœud $v \in X$: $d(v) = \infty$, $p(v) = null$ et $S(v) = unreached$. La première étape met à jour les informations sur le nœud racine : $d(s) = 0$ et $S(s) = labeled$. Ensuite, l'opération SCAN est appelée jusqu'à ce que tous les nœuds soient *scanned*. L'opération SCAN change éventuellement le statut *unreached* ou *scanned* des nœuds en *labeled*. En l'absence de cycle de coût négatif, la méthode de marquage s'achève et les parents définissent un arbre des plus courts chemins alors que $d(v)$ est la longueur du plus court chemin de s à v pour chaque nœud v.

Function SCAN(v)

```
for each (v, w) ∈ A do
    if d(v) + l(v, w) < d(w) then
        d(w) ← d(v) + l(v, w) ;
        S(w) ← labeled ;
        p(w) ← v ;
S(v) ← scanned ;
```

L'algorithme de Bellman-Ford maintient l'ensemble des noeuds marqués dans une queue de priorité FIFO : le prochain nœud à examiner est pris en tête alors qu'un nœud nouvellement marqué est ajouté en queue. L'algorithme de Dijkstra sélectionne le nœud marqué v dont la valeur $d(v)$ est minimale, mais ne peut traiter que des longueurs positives.

Propagation de contraintes D'un autre côté, la PPC associe un algorithme de filtrage FILTER(C) à chaque contrainte C, qui réduit les domaines des variables en supprimant les valeurs inconsistantes, *i.e.* qui ne peuvent pas appartenir à une solution de la contrainte. FILTER(C) renvoie l'ensemble des variables qui ont été modifiées durant l'étape de filtrage. Après chaque modification d'un domaine d'une variable, toutes les contraintes impliquant cette variable doivent être révisées à nouveau puisque de nouvelles réductions de domaine sont possibles. Ce processus est répété jusqu'à ce qu'aucune réduction ne soit plus possible ce qui arrive nécessairement puisque les domaines sont finis. La fonction FILTER(Q) met en œuvre le mécanisme appelé *propagation de contraintes*.

Function FILTER(Q)

```
while Q ≠ ∅ do
    /* pick y in Q and remove y from Q                */
    for each constraint C involving y do
        M ← FILTER(C) ;
        if ∃x ∈ M/D(x) = ∅ then return false;
        Q ← Q ∪ M;
return true;
```

Les conclusions de plusieurs études [8, 3, 1] sur l'ordonnancement des révisions de variables/contraintes pour la consistance d'arc sont les suivantes. Une *propagation orientée par les variables* est préférable à une propagation orientée par les contraintes. La meilleure stratégie consiste à *sélectionner une variable dont la taille du domaine est minimale*. La propagation orientée variable sélectionne successivement les variables dont le domaine a été réduit et appelle les algorithmes de filtrages des contraintes impliquant cette variable. La propagation orientée par les contraintes considère les contraintes à tour de rôle sans se soucier des variables, comme l'algorithme AC-3. Les conclusions sont similaires pour les méthodes de marquage [7].

3 Modèle de plus court chemin

Nous décrivons maintenant un modèle PPC simple pour le problème de plus court chemin. Pour chaque nœud $v \in X$, soit $d(v)$ une variable entière positive ou nulle sauf pour le nœud racine où $d(s) = 0$. Pour chaque arc $(u, v) \in A$, nous posons une contrainte $d(v) \leq d(u) + l(u, v)$ qui réalise la *consistance de borne* plutôt que la consistance d'arc. Par conséquent, la borne supérieure du domaine de $d(v)$ après la propagation initiale est la plus courte distance d'un chemin partant de s (pas de recherche arborescente).
À première vue, nous pouvons simuler l'algorithme de Bellman-Ford en utilisant une queue FIFO Q et l'algorithme de Dijkstra en choisissant une variable de Q dont la taille du domaine est minimale (les bornes inférieures des domaines sont 0 et les bornes supérieures sont les distances depuis s). En réalité, cette affirmation est fausse pour presque tous les solveurs existants à cause de l'initialisation des contraintes, *i.e.* le premier appel à leurs algorithmes de filtrage. Nous allons voir que la plupart des solveurs ajoutent une nouvelle contrainte uniquement lorsque la propagation des contraintes du sous-problème courant est terminée.

4 Modification de la propagation

Initialisation des contraintes La plupart des solveurs gèrent uniquement les modifications entre deux appels consécutifs du même algorithme de filtrage. Cette information peut être exploitée par les algorithmes de filtrage, mais son maintien est compliqué par la propagation des autres contraintes/variables qui a eu lieu entre la modification du domaine et la propagation d'une variable. FILTER(C) utilise généralement les domaines courants quand la contrainte est initialisée, mais le filtrage de la contrainte C peut alors être appelé plusieurs fois pour les mêmes modifications si plusieurs variables de C sont dans Q. Dans ce cas, le solveur applique FILTER(C) pour des modifications

antérieures à l'initialisation de la contrainte, ce qui est déraisonnable. Une solution simple adoptée dans la plupart des solveurs consiste donc à retarder l'initialisation d'une contrainte tant que le sous-problème courant n'a pas été entièrement propagé.
Cette solution a de sérieuses conséquences illustrées sur le PCC décrit en figure 1 où les arcs sont étiquetés par leur longueur et leur contrainte. Un solveur ajoutera les contraintes à tour de rôle et propagera les modifications sur le sous-problème courant. Les contraintes seront ajoutées ou filtrées dans l'ordre suivant : $C_1, C_2, C_3, C_4, C_5, C_6$ (v_1 est réduite), C_2 (v_2 est réduite), C_4, C_7, C_8, C_9 (v_1 est réduite), C_2 (v_2 est réduite), C_4, C_{10}, C_{11}, C_{12} (v_1 est réduite), C_2, et C_4. La méthode de marquage insérera s dans la queue. Les contraintes $C_1, C_3, C_5, C_8, C_{11}$ seront traitées depuis s et ajouterons les nœuds v_1, v_2, v_3, v_4, v_5 dans la queue. Ensuite, v_1 est extraite ; C_2 est filtrée ; v_2 est extraite ; C_4 est filtrée ; v_3 est extraite ; C_6 est filtrée (ajoute v_1) ; C_7, C_9, C_{10}, C_{12} sont filtrées ; v_1 est extraite ; C_2 est filtrée (ajoute v_2) ; v_2 est extraite et C_4 est filtrée. La méthode de marquage examine moins de contraintes/arcs plusieurs fois : les contraintes C_2 et C_4 ne sont filtrées que deux fois au lieu de quatre.

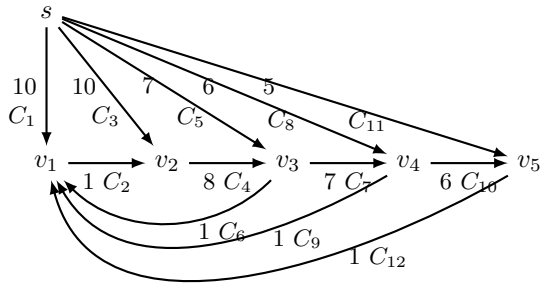


FIGURE 1 – Un exemple de problème de plus court chemin.

Pour résoudre ce problème, nous avons séparé l'initialisation de la propagation ce qui permet d'ajouter de nouvelles contraintes sans attendre la fin de la propagation du sous-problème courant. Nous dirons que la propagation est immédiate quand la fonction FILTER(Q) est appelée dès qu'une modification a eu lieu.

Heuristique des parents L'algorithme de Bellman-Ford peut être amélioré en utilisant l'heuristique des parents (*parent checking*) qui examine un nœud v si son parent $p(v)$ n'appartient pas à Q. Dans un problème de plus court chemin, la présence du parent d'un nœud v dans la queue Q entraînera nécessairement une nouvelle modification de $d(v)$. Même si cela n'est pas nécessairement le cas pour la propagation de contraintes, nous évaluerons une nouvelle variante où la propagation d'une variable est retardée tant que son parent appartient à la queue Q.

5 Expérimentations

Nous récapitulons les résultats obtenus pour plusieurs variantes de la propagation de contraintes implantées dans le solveur Choco (`http://choco.mines-nantes.fr`). Soit P** et _** les variantes avec et sans vérification des parents. Soit *F* et *D* les variantes où la sélection de la prochaine variable suit une politique FIFO ou du domaine minimal. Soit **_ et **S les variantes avec propagation immédiate ou initialisation séparée. La sélection d'une variable a une complexité au plus linéaire en fonction de $|Q|$.

Plus court chemin Les résultats sur les problèmes de plus court chemin donnés dans le tableau (a) concernent quatre catégories de graphes générés aléatoirement contenant 2000 nœuds et approximativement 2 millions de contraintes. Les catégories D et SD définissent des graphes orientés acyclique complet ou semi-complet avec des longueurs positives et 20% d'arcs supplémentaires. Dans un graphe semi-complet, les degrés entrants et sortants de la moitié des nœuds sont unitaires. Chaque arc supplémentaire crée un cycle de longueur positive. Les catégories DN et SDN se distinguent de D et SD par la présence d'arcs de longueur négative, mais ne créant aucun cycle de longueur négative. Pour chaque catégorie, les lignes $\#v$ et t correspondent au nombre de variables propagées et à la durée de propagation en secondes.
La politique du domaine minimal est plus efficace que la politique FIFO et l'initialisation séparée se révèle intéressante. La combinaison des deux simule parfaitement l'algorithme de Dijkstra même en présence de longueurs négatives. Par conséquent, la propagation des variables est « minimale » puisque chaque variable est propagée une seule et unique fois (toutes les variables sont modifiées). Par contre, la politique du domaine minimal est plus lente que FIFO probablement à cause de la taille de Q sauf pour la catégorie DN.
L'heuristique des parents offre une légère amélioration par rapport à *F_ en retardant approximativement la propagation de 1% des variables. Par contre, elle ralentit significativement la propagation de *FS parce que 135% des variables de Q sont retardées (certaines variables sont retardées plusieurs fois) et Q est de grande taille. Nous avons aussi évalué des variantes de l'heuristique des parents (un ancêtre est dans la queue, plusieurs parents ...) sans noter d'amélioration particulière. En conclusion, ces résultats montrent que l'heuristique des parents a un intérêt limité pour la PPC.

Affectation de fréquences radio Les modèles du RLFAP sont constitués de variables (6000 en moyenne) avec des domaines énumérés et de contraintes arith-

		_F_	_FS	_D_	_DS	PF_	PFS	PD_	PDS
D	$\#v$	10516	7551	9164	1999	10343	7480	9164	1999
	t	16.4	3.2	14.7	4.8	16.3	12.2	3.4	5.6
SD	$\#v$	10231	7888	9164	1999	10091	7772	9164	1999
	t	6.9	1.8	6.4	2.7	6.8	5.5	1.9	3.3
DN	$\#v$	13916	7177	12506	1999	13732	7146	12506	1999
	t	19.1	12.0	17.1	6.0	18.8	13.5	16.1	6.7
SDN	$\#v$	13021	7328	11885	1999	12867	7291	11885	1999
	t	7.6	1.7	7.1	2.7	7.6	5.2	2.1	3.4

(a) Résultats pour les problèmes de plus court chemin.

	_F_	_FS	_D_	_DS
$\%v$	101	92	99	84
$\%r$	5	2	4	1
t	0.67	0.78	0.70	3.30

	_F_		_D_	
	$\%\!\circ v$	$\%\!\circ r$	$\%\!\circ v$	$\%\!\circ r$
SAT	30.1	1.5	28.6	0.1
UNSAT	4.6	0.5	1.9	0.1

(b) RLFAP : Propagation initiale et shaving

métiques imposant la consistance d'arc ou de borne. Le tableau supérieur (b) récapitule les résultats de la propagation initiale. Les variantes avec l'heuristique des parents n'y apparaissent pas car elles n'apportent aucune amélioration même si 13% et 89% des variables propagées sont retardées pour PF_ et PFS. Les lignes $\%v$, $\%r$ et t représentent respectivement le pourcentage de variables propagées par rapport au nombre total de variables, le pourcentage de variables ré-entrantes par rapport au nombre de variables propagées et la durée de propagation en secondes. **S réduit le nombre de variables propagées et ré-entrantes sans toutefois améliorer la durée de propagation surtout pour _DS. La propagation peut être améliorée en réduisant le nombre de fois où une même variable est propagée. Chaque variable modifiée doit être extraite au moins une fois de la queue Q puisqu'il est nécessaire d'étudier les conséquences de sa modification. Donc, une « propagation parfaite » ne propagerait au mieux qu'une seule fois chaque variable modifiée. Ainsi, *le nombre de variable ré-entrantes est une borne supérieure sur la diminution du nombre de variables traitées pendant une phase de propagation par une autre politique d'ordonnancement.*

Le tableau inférieur (b) donne les résultats d'une phase de *shaving* sur les bornes des domaines. **S n'y apparaît pas puisque les contraintes ont déjà été initialisées. Les lignes $\%\!\circ v$ et $\%\!\circ r$ correspondent respectivement au pour-mille des variables propagées et ré-entrantes. Les temps de calcul ne sont pas mentionnés, car le solveur a été lourdement instrumenté. Les résultats sont regroupés selon la réponse donnée par la propagation (550000 SAT, 130000 UNSAT). Les réponses SAT où une seule variable est propagée ont été ignorées. Le nombre de variables propagées est légèrement réduit grâce à _D_, surtout pour les réponses UNSAT. *Le nombre de variables ré-entrantes est très faible ce qui limite l'amélioration espérée en utilisant un autre ordonnancement des variables.* Ces résultats confirment les observations de [4] : *moins de variables sont propagées pour une réponse UNSAT que pour SAT.*

6 Conclusion

Nous avons montré que les solveurs doivent être modifiés pour être compétitifs avec les meilleurs algorithmes pour résoudre les problèmes de plus court chemin. Nous avons modifié le mécanisme de propagation de contraintes pour le rendre plus efficace sur des problèmes tels que PCC ou RLFAP. Malheureusement, nous avons aussi montré qu'un nombre restreint de variables sont traitées plusieurs fois pendant une phase de propagation ce qui limite l'amélioration induite par un autre ordonnancement des variables.

Références

[1] T. Balafoutis and K. Stergiou. Exploiting constraint weights for revision ordering in AC algorithms. In *Proc. of ECAI-2008-W31*, 2008.

[2] R. Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16 :87–90, 1958.

[3] F. Boussemart, F. Hemery, and C. Lecoutre. Revision ordering heuristics for the CSP. In *Proc. of CPAI 2004*, pages 29–43, 2004.

[4] F. Boussemart, F. Hemery, C. Lecoutre, and M. Samy-modeliar. Contrôle statistique du processus de propagation de contraintes. In *Proc. of JFPC 2011*, pages 65–74, 2011.

[5] B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest paths algorithms : Theory and experimental evaluation. *Math. Program.*, 73 :129–174, 1996.

[6] E.W. Dijkstra. A note on two problems in connection with graphs. *Num. Math.*, 1 :269–271, 1959.

[7] Y. Dinitz and R. Itzhak. Hybrid bellman-ford-dijkstra algorithm. Technical Report CS-10-04, Ben-Gurion University of the Negev, 2010.

[8] R.J. Wallace and E.C. Freuder. Ordering heuristics for arc consistency algorithms. In *Proc. of Canadian AI 1992*, pages 163–169, 1992.

Une heuristique de génération de colonnes pour le problème de tournées de véhicules avec faisabilité boîte noire Extended Abstract *

Florence Massen[1] **Yves Deville**[1] **Pascal Van Hentenryck**[2]

[1] ICTEAM, Université catholique de Louvain, Belgium
[2] Optimization Research Group, NICTA, University of Melbourne, Australia
{Florence.Massen, Yves.Deville}@uclouvain.be pvh@nicta.com.au

1 Introduction

Une grande attention a été accordée aux problèmes de tournées de véhicules (VRP) depuis le début des années 1960. Initialement, seules des variantes de base de ce problème ont été prises en compte. Ensuite, la recherche a porté sur des variantes plus complexes, tels que des problèmes avec fenêtres de temps ou avec collecte et livraison. Ces dernières années les problèmes de tournées de véhicules riches ont été analysé. Ceux-ci visent à donner une représentation plus réaliste des problèmes rencontrés dans le monde réel. Dans les problèmes riches il est souvent nécessaire de combiner différentes contraintes complexes qui n'avaient été considérées qu'individuellement dans la littérature.

Pour aborder les problèmes riches, des méthodes de recherches locales ainsi que des méthodes issues de la programmation mathématique existent. La programmation par contraintes est parfois utilisée en combinaison avec d'autres méthodes d'optimisation pour ce genre de problèmes.

Les problèmes riches sont souvent traités en adaptant des méthodes existantes au contraintes spécifiques considérées (comme par exemple dans la génération de colonnes proposée dans [1] où la procédure de pricing est adaptée aux contraintes du problème).

Dans le contexte des problèmes riches, un nouveau type de VRP a vu le jour, les problèmes de tournées combinés avec d'autres problèmes combinatoires (p.ex. combinaison du VRP avec problèmes de chargement (3L-CVRP [9]) ou problème de planification (VRPTW with Driver Scheduling [12])). Ces problèmes sont souvent abordés en utilisant des approches très dédiées. Le but de cet article est de proposer une reformulation généralisée pour ce type de problème ainsi qu'une procédure d'optimisation pour le problème générique résultant de cette reformulation. Pour ce faire, nous introduisons le VRP avec faisabilité boîte noire (VRPBB). Ce problème est une extension du VRP de base. Outre le respect des contraintes VRP (capacité, fenêtres de temps, ...), chaque tournée doit vérifier un ensemble de contraintes inconnues F. Il est donc impossible d'accéder aux contraintes non-VRP à vérifier par chaque tournée. La faisabilité d'un itinéraire par rapport à F est vérifiée à l'aide d'un algorithme boîte noire fourni.

Nous proposons de reformuler le VRPBB comme un problème de partitionnement d'ensembles que nous optimisons à l'aide d'une approche (heuristique) de génération de colonnes. Des fourmis collectrices génèrent et collectionnent des colonnes (tournées) de façon heuristique, tout en étant guidées par des dépôts de phéromones provenant d'un oracle externe. Cet oracle calcule les dépôts de phéromones en fonction de la solution actuelle à la relaxation du problème. L'approche nous permet d'améliorer de manière itérative la borne inférieure du problème considéré. Une solution entière est trouvée en résolvant le problème de partitionnement d'ensembles avec solveur MIP.

Cet article présente trois contributions. D'abord, nous proposons un nouveau problème générique, le problème de tournées de véhicules avec faisabilité boîte noire, qui permet de représenter des problèmes de tournées de véhicules exigeant la résolution d'un problème combinatoire

*Ce papier est une version courte de [7]

par tournée. Deuxièmement, nous proposons un algorithme pour résoudre ce problème générique. Notre méthode est donc indépendante du problème combinatoire à résoudre pour chaque tournée. Elle peut facilement être appliquée à un nouveau problème en utilisant une autre fonction de faisabilité boîte noire. Enfin, nous démontrons l'applicabilité de la méthode proposée sur deux problèmes, le VRP avec chargement en trois dimensions (3L-CVRP) et le VRP à piles multiples (MP-VRP). Nous comparons nos résultats avec ceux des approches dédiées existantes pour montrer que notre approche générique est très compétitive.

1.1 Travaux apparentés

L'optimisation sous la présence de fonctions boîte noire est un domaine de recherche actif. Des métaheuristiques classiques tels que les algorithmes génétiques ou le recuit simulé ont été conçus en tant qu'algorithmes d'optimisation boîte noire [11]. Des fonctions boîte noire coûteuses à évaluer peuvent être trouvées dans des problèmes d'optimisation structurelle, telle que dans la conception de bateaux ou la conception de zones de compression pour voitures, où des simulations sont nécessaires pour juger de la qualité d'une solution. L'évaluation des fonctions boîte noire considérées dans ce domaine peut prendre jusqu'à vingt heures [10]. Dans VRPBB, nous supposons simplement que l'évaluation de la faisabilité est coûteuse en comparaison avec des variantes typiques du VRP.

La littérature se concentrant explicitement sur les problèmes de tournées de véhicules avec évaluation de faisabilité coûteuse est rare. Les auteurs ont connaissance d'un seul papier ([14]) où on y évalue l'efficacité de différentes structures d'indexation pour stocker l'information de faisabilité dans le cas d'un VRP combiné avec un problème de chargement à deux dimensions. Dans [3] les auteurs se concentrent sur les VRP avec des graphes de faisabilité creux.

Finalement l'utilisation d'heuristiques en combinaison avec la génération de colonnes est bien connue. La littérature portant sur la génération de colonnes combinée avec des colonies de fourmis est moins dense. Un exemple est [2] où la génération de colonnes est utilisée afin de générer une solution qui permet d'initialiser les pistes de phéromones pour les fourmis.

2 Le problème de tournées de véhicules avec faisabilité boîte noire

Problème de tournées véhicules avec contraintes de capacité Le problème de tournées véhicules avec contraintes de capacité (CVRP, [16]) est sous-jacent à la plupart des variantes VRP. Il est défini sur un graphe complet et pondéré $G = (V,E)$ où $V = \{0,1,...,n\}$ est un ensemble de $n+1$ sommets et E l'ensemble des arêtes pondérées reliant chaque paire de sommets.

Le sommet 0 représente le dépôt tandis que les sommets $1,\ldots,n$ sont les n clients attendant d'être servis. Le poids non-négatif c_{ij} $(i,j = 0,...,n : i \neq j)$ d'une arête (i,j) correspond au coût du voyage du sommet i au sommet j. La flotte homogène est limitée à K véhicules, chacun ayant une capacité maximale D. Avec chaque client i $(i = 1,...,n)$ est associée une demande d_i. Une tournée r est définie par l'ensemble des sommets clients visités E et la séquence σ dans laquelle ces sommets sont visités. Chaque tournée commence et se termine au dépôt. La tournée r peut être décrite par (S,σ).
Enfin, le but est de mettre au point une solution composée d'au plus K routes de telle sorte que :

- chaque client soit visité sur une tournée exactement,
- la somme des demandes des clients sur une tournée ne dépasse pas la capacité maximale D,
- le coût total des déplacements, égal à la somme des poids des arêtes traversées, soit minimisé.

Problème de tournées de véhicules avec faisabilité boîte noire Dans le VRPBB chaque tournée faisable doit vérifier et les contraintes liées à la variante VRP sous-jacente et un ensemble de contraintes dures inconnues appelé F.

Supposons que $feas(r,c) = true$ indique que la tournée r satisfait la contrainte $c \in F$. Une tournée provisoire r est considérée comme faisable par rapport à F si et seulement si $\bigwedge_{c \in F} feas(r,c)$.

La boîte noire fournit une fonction déterministe retournant un booléen indiquant la faisabilité de la tournée r par rapport à F. Cette fonction est considérée comme coûteuse en ressources informatiques (par rapport aux fonctions de faisabilité couramment rencontrés dans le VRP) et plus précisément de complexité non-linéaire dans la longueur de la route. Une tournée respectant les contraintes CVRP est appelée VRP-faisable. Une tournée respectant toutes les contraintes dans F est appelée BB-faisable. Finalement une tournée qui est en même temps VRP- et BB-faisable est appelée faisable.

3 Approche choisie

Nous ne cherchons pas à trouver un optimum global pour le VRPBB. La recherche locale en combinaison avec des bonnes fonctions de voisinage a montré son efficacité sur de nombreux problèmes. Toutefois, une approche de recherche locale ne semble pas appropriée pour le VRPBB. Compte tenu de l'ensemble des contraintes inconnues F nous ne pouvons pas être certain que le voisinage faisable pour un VRPBB donné soit connexe. En outre, nous ne pouvons pas extraire une mesure de violation de la boîte noire. Cela ne nous permet donc pas de visiter des solutions infaisables avec une pénalité basée sur le nombre de

violations. Il est donc difficile d'appliquer une approche métaheuristique. C'est pourquoi nous avons décidé de reformuler le VRPBB comme un problème de partitionnement d'ensembles (SPP).

Soit $\mathscr{R}$ l'ensemble des tournées possibles, c_r le coût du tournée r et x_r une variable indiquant si le tournée r est utilisé dans la solution optimale. Le SPP analogue au problème de tournées de véhicules est défini comme suit :

$$
\begin{aligned}
Min \quad & \sum_{r\in\mathscr{R}} c_r x_r \\
s.t. \quad & \sum_{r\in\mathscr{R}} v_{ir} x_r = 1 \quad \forall i \in V\backslash 0 \\
& \sum_{r\in\mathscr{R}} x_r \leq K \\
& x_r \in \{0,1\} \quad \forall r \in \mathscr{R}
\end{aligned}
$$

$$
v_{ir} = \begin{cases} 1 & \text{si client } i \text{ visité dans tournée } r \\ 0 & \text{sinon} \end{cases}
$$

$\forall i \in V\backslash 0, \forall r \in \mathscr{R}$

Le SPP est défini sur l'ensemble de toutes les tournées possibles $\mathscr{R}$. Le calcul de $\mathscr{R}$ est évidemment irréaliste si ce n'est pour les problèmes de taille réduite. Une approche bien connue pour ce type de problèmes est la génération de colonnes. L'idée est de travailler avec la relaxation linéaire du problème initial et d'ajouter itérativement des nouvelles colonnes à la formulation du problème relâché. On travaille ainsi sur un problème restreint, appelé le problème maître restreint (RMP). Les nouvelles colonnes sont découvertes par la résolution d'un sous-problème. De nouvelles colonnes intéressantes sont identifiées à l'aide de leur coût réduit, qui est obtenu en utilisant les coûts découlant du problème dual du RMP courant (pricing). Typiquement, seules les colonnes de coût réduit négatif (dans le cas d'un problème de minimisation) sont ajoutées à la formulation du problème. Dans notre cas, générer des colonnes revient à générer des tournées faisables.

3.1 Une heuristique de génération de colonnes

Puisque nous ne cherchons pas une solution exacte à notre problème, nous allons utiliser une approche heuristique de génération de colonnes. À chaque itération, nous générons un certain nombre de tournées faisables qui seront ajoutées à la formulation du problème relâché. Ces routes sont générées en utilisant des fourmis collectrices, expliquées à la section suivante. Ce sont ces fourmis collectrices qui assureront la faisabilité (VRP- et BB-faisabilité) des tournées générées. Le RMP courant est alors résolu de manière optimale. Cela fournit les coûts duaux permettant de calculer le coût réduit de nouvelles tournées. Enfin nous mettons à jour la matrice de phéromones en nous basant sur la solution au problème relâché actuel. Finalement une solution entière est trouvée en résolvant le SPP sur l'ensemble des tournées accumulées (en utilisant un solveur MIP). Pour plus de détails ainsi que l'algorithme complet utilisé veuillez-vous référer à [7].

3.2 Les fourmis collectrices

Nous proposons des exécutions répétées d'une heuristique aléatoire appelée fourmi collectrice pour générer des tournées faisables. Les fourmis collectrices sont basées sur les fourmis "savings-based" de [13]. Les auteurs de ce papiers se sont inspirés de l'heuristique "savings" [5]. Dans notre contexte, une approche basée sur les colonies de fourmis est un choix naturel en raison de la facilité avec laquelle le guidage par la solution actuelle relâchée peut être implémenté en utilisant des phéromones. Pour plus de détails ainsi que l'algorithme complet utilisé veuillez-vous référer à [7].

3.3 Mise à jour des phéromones

Chaque fourmi dispose de la matrice de phéromones qui indique la quantité de phéromones sur chaque arête (i, j). Les phéromones influencent l'attractivité d'une arête, et donc la probabilité d'être incluse dans une tournée par une fourmi. L'idée derrière les phéromones est de guider les fourmis vers des tournées de bonne qualité et avec une plus grande probabilité d'être BB-faisables. Comme les tournées de la solution courante sont optimales pour le RMP courant et qu'en plus nous savons que ces tournées sont VRP- et BB-faisables, nous aimerions que les fourmis produisent des tournées leurs ressemblant. Ceci se fait dans l'espoir de produire des tournées de qualités similaires. Une fois que la solution actuelle relâchée a été calculée, la matrice de phéromones est mise à jour en évaporant une partie des phéromones actuelles et en déposant une nouvelle quantité de phéromones apparaissant dans la solution actuelle. La quantité déposée sur une arête dépend du nombre de fois que celle-ci apparaît dans la solution actuelle.

4 Résultats expérimentaux

Notre approche a été implémentée en Comet (utilisant CLP pour solveur LP et SCIP en solveur MIP) et testée sur deux problèmes différents combinant tournées et chargement. Pour la description exacte des paramètres et conditions d'exécution veuillez vous référer à [7].

Dans le 3L-CVRP ([9]) la demande de chaque client est exprimée par un ensemble d'objets rectangulaires à trois dimensions. Le volume de chargement des camions est limitée en trois dimensions. Pour qu'une tournée soit faisable il faut qu'un chargement faisable de tous les objets

associés aux clients visités existe. Au delà des contraintes de bin packing à trois dimensions, des contraintes complexes telles que le support, la fragilité et l'accessibilité des objets dans les camions doivent aussi être considérées. Nous comparons notre approche à quatre approches dédiées existantes. Dans [9] les auteurs proposent une recherche tabu (TS) dans laquelle ils permettent de visiter des solutions infaisables. Une recherche tabu guidée (GTS) a été proposée par [15], tandis qu'une approche par colonie de fourmis (ACO) a été présentée dans [8]. Finalement [4] présente une recherche tabu (HTS). Toutes ces approches sont dédiées (certaines plus que d'autres) au problème sous considération. Elles utilisent des heuristiques ou métaheuristiques pour le problème de chargement.

Pour tester notre approche sur le 3L-CVRP nous avons utilisés les instances disponibles sur `http://www.or.deis.unibo.it/research.html`. La génération de ces instances est expliquée dans [9]. En tant que fonction de faisabilité boîte noire nous avons réimplémenté (en C++) l'approche heuristique décrite dans [4] (HTS). Nous utilisons donc une heuristique de chargement différent de ACO, GTS et TS. La génération de nouvelles tournées est arrêtée soit quand le temps limite est écoulé (même limites qu'en [9]), soit quand la borne inférieure n'a pu être améliorée pendant un nombre fixé d'itérations.

Les résultats montrent qu'en termes de qualité des solutions notre approche générique est très compétitive avec HTS et ACO (approches dédiées) tandis qu'elle améliore les résultats trouvés par GTS et TS (approches dédiées). En ce qui concerne les temps d'exécution notre approche est comparable à TS (mêmes limites de temps) et légèrement moins bonne qu'ACO et GTS. En ce qui concerne les temps d'exécution atteints par HTS, ils sont nettement inférieurs à ceux de ACO, GTS et TS, et donc aussi aux nôtres. Pour une liste et comparaison exhaustive des résultats sur le 3L-CVRP veuillez voir [7].

Le second problème sur lequel notre approche a été testée est le MP-VRP [6]. Dans ce problème les clients demandent des objets à deux dimensions. La longueur du camion est divisé en trois piles de même largeur. Les objets demandés par les clients peuvent prendre n'importe quelle hauteur, mais uniquement une longueur correspondant soit à la largeur d'une pile, ou à la largeur de 3 piles. En plus des contraintes du bin packing à deux dimensions le chargement doit être tel que lors de la visite d'un client, tous les objets qu'il demande sont au-dessus des piles.

Nous comparons notre approche à trois approches dédiées existantes. Dans [6] Doerner et al. proposent une approche colonie de fourmis (ACO) et une approche recherche tabu (TS) qui sont analogues à celles proposées dans [8] et [9] pour le 3L-CVRP. Dans [17] les auteurs proposent une descente à voisinage variable. Ils permettent l'infaisabilité des solutions visitées. L'ACO et le TS utilisent une heuristique pour le chargement, tandis que dans le VNS on exploite le fait de disposer d'une heuristique et d'une méthode exacte.

Pour tester notre approche sur le MP-VRP nous avons utilisé les instances disponibles sur `http://prolog.univie.ac.at/research/VRPandBPP/`. La génération de ces instances est expliquée dans [6]. Les auteurs de [17] ont eu l'amabilité de nous fournir leur implémentation du test de faisabilité pour le chargement. Nous n'avons pas effectué de changements mais seulement imposé une limite de temps de 5 secondes sur la méthode exacte. De nouveau, la génération de nouvelles routes est arrêtée soit quand le temps limite est écoulé (même limite que [17]), soit quand la borne inférieure n'a pas pu être améliorée pendant un nombre fixé d'itérations.

Notre approche générique ne permet pas d'atteindre exactement la même qualité de solutions que les approches dédiées existantes. La recherche s'arrête que rarement avant l'écoulement de la limite de temps. Malheureusement nous ne disposons pas des temps d'exécution exactes pour le VNS. Pour des instances de plus grande taille, ACO et TS utilisent des limites de temps différentes et atteignent un temps d'exécution total nettement supérieur au nôtre.

En résumé, notre approche générique, comparée à des approches dédiées, permet de trouver des résultats très compétitifs pour le 3L-CVRP et légèrement moins bons pour le MP-VRP. Le temps d'exécution total supérieur est du à la généricité de notre approche. Cette perte de temps, ainsi que la légère détérioration de la qualité des solutions pour le MP-VRP est compensée par le gain en temps en utilisant notre approche générique plutôt que de développer une approche dédiée.

5 Conclusion

Cet article présente le nouveau problème générique de tournées de véhicules avec faisabilité boîte noire (VRPBB). Dans le VRPBB chaque tournée doit vérifier un ensemble de contraintes dures et inconnues. La faisabilité d'une tournée ne peut être vérifiée qu'en utilisant un algorithme boîte noire coûteux. La difficulté de ce problème provient essentiellement de la structure du problème inconnu, mais aussi de la vérification de faisabilité coûteuse d'une tournée. Pour résoudre le VRPBB nous proposons une approche basée sur la génération de colonnes. Des fourmis collectrices produisent et collectionnent des tournées avec le potentiel d'augmenter la qualité de la solution. Ces fourmis sont guidées par des dépôts de phéromones provenant de la solution au problème relâché courant. Une solution entière est finalement trouvée en résolvant un problème de partitionnement d'ensembles sur les tournées recueillies. Nous avons montré l'applicabilité de la méthode proposée sur deux applications pratiques, le problème de tournées de véhicules avec chargement en trois dimensions et le problème de tournées de véhicules à piles multiples. Nous

montrons que notre approche générique est en concurrence avec des approches spécifiques, cela au prix d'un temps d'exécution un peu plus élevé. L'effort de développement nécessaire d'utiliser notre approche est nettement inférieur à l'effort nécessaire pour construire une approche dédiée. Les travaux futurs consisteront à inclure notre approche en tant qu'étape de pricing dans un framework Branch-and-Price heuristique. Nous allons également tester l'approche proposée sur d'autres applications telles que le VRP avec fenêtres de temps et planification des pauses de conducteurs.

Références

[1] Ceselli A., Righini G., and Salani M. A column generation algorithm for a rich vehicle-routing problem. *Transport. Sci.*, 43, 2009.

[2] Merel A., Gandibleux X., and Demassey S. A collaborative combination between column generation and ant colony optimization for solving set packing problems. In *9th Metaheuristics International Conference (MIC 2011)*, Udine, Italy, 2011.

[3] J.E. Beasley and N. Christofides. Vehicle routing with a sparse feasibility graph. *Eur. J. Oper. Res.*, 98, 1997.

[4] A. Bortfeldt. A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. Technical report, FernUniversität in Hagen, 2010.

[5] G. Clarke and J.V. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Oper. Res.*, 12, 1964.

[6] K.F. Doerner, G. Fuellerer, R.F. Hartl, M. Gronalt, and M. Iori. Metaheuristics for the vehicle routing problem with loading constraints. *Networks*, 49, 2007.

[7] Massen F., Deville Y., and Van Hentenryck P. Pheromone-based heuristic column generation for vehicle routing problems with black box feasibility. In *CPAIOR'12*, volume 7298 of *LNCS*, Nantes, France, 2012.

[8] G. Fuellerer, K. F. Doerner, R.F. Hartl, and M. Iori. Metaheuristics for vehicle routing problems with three-dimensional loading constraints. *Eur. J. Oper. Res.*, 201, 2010.

[9] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search algorithm for a routing and container loading problem. *Transport. Sci.*, 40, 2006.

[10] D.R. Jones, M. Schonlau, and W.J. Welch. Efficient global optimization of expensive black-box functions. *J. Global Opt.*, 13, 1998.

[11] M. Laguna, J. Molina, F. Pérez, R. Caballero, and A. G-Hernández-Díaz. The challenge of optimizing expensive black boxes : a scatter search/rough set theory approach. *J. Oper. Res. Soc.*, 61, 2010.

[12] E. Prescott-Gagnon, G. Desaulniers, M. Drexl, and L.-M. Rousseau. European driver rules in vehicle routing with time windows. *Transport. Sci.*, 44, 2010.

[13] M. Reimann, M. Stummer, and K.F. Doerner. A savings based ant system for the vehicle routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002.

[14] J. Strodl, K. Doerner, F. Tricoire, and R. Hartl. On index structures in hybrid metaheuristics for routing problems with hard feasibility checks : An application to the 2-dimensional loading vehicle routing problem. In *Hybrid Metaheuristics*. 2010.

[15] C.D. Tarantilis, E.E. Zachariadis, and C.T. Kiranoudis. A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. *IEEE Trans. Intell. Transp. Syst.*, 10, 2009.

[16] P. Toth and D.Vigo, editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, 2002.

[17] Fabien Tricoire, Karl F. Doerner, Richard F. Hartl, and Manuel Iori. Heuristic and exact algorithms for the multi-pile vehicle routing problem. *OR Spectrum*, 2009.

Un nouvel algorithme de consistance locale sur les nombres flottants

Mohammed Said BELAID, Claude MICHEL, Michel RUEHER

I3S UNS/CNRS, 2000 route des Lucioles, BP 121, 06903 Sophia Antipolis Cedex, France
{MSBelaid, Claude.Michel}@i3s.unice.fr, Michel.Rueher@gmail.com

Résumé

La résolution de contraintes sur les nombres à virgule flottante soulève des problèmes critiques dans de nombreuses applications, notamment en vérification de programmes. Jusqu'à maintenant, les algorithmes de filtrage sur les nombres à virgule flottante ont été limités à la 2B–consistance et ses dérivées. Bien que ces filtrages soient conservatifs des solutions, ils souffrent des problèmes bien connus des consistances locales, e.g., leur incapacité à traiter efficacement les occurrences multiples de variables. Leurs limitations proviennent aussi de la pauvreté des propriétés de l'arithmétique des nombres à virgule flottante. Afin de pallier à ces limitations, nous proposons dans cet article un nouvel algorithme de filtrage de contraintes sur les flottants qui repose sur des relaxations successives sur les réels du problème initial sur les flottants. Des bornes conservatives des domaines sont obtenues à l'aide d'un solveur de programme linéaire mixte (MILP) appliquées à des linéarisations conservatives de ces relaxations. Les résultats préliminaires sont prometteurs et montrent que cette approche peut effectivement accélérer les filtrages par consistances locales.

Abstract

Solving constraints over floating-point numbers is a critical issue in numerous applications notably in program verification. Capabilities of filtering algorithms over the floating-point numbers have been so far limited to 2b-consistency and its derivatives. Though safe, such filtering techniques suffer from the well known pathological problems of local consistencies, e.g., inability to efficiently handle multiple occurrences of the variables. These limitations also take roots in the strongly restricted floating-point arithmetic. To circumvent the poor properties of floating-point arithmetic, we propose in this paper a new filtering algorithm which relies on various relaxations over the reals of the problem over the floats. Safe bounds of the domains are computed with a mixed integer linear programming solver (MILP) on safe linearizations of these relaxations. Preliminary experiments on a relevant set of benchmarks are very promising and show that this approach can be very effective for boosting local consistency algorithms over the floats.

1 Introduction

Les logiciels critiques s'appuient de plus en plus sur le calcul en virgule flottante. Par exemple, les systèmes embarqués sont contrôlés par des logiciels qui utilisent des mesures et des données de leur environnement évaluées par des nombres flottants. Les calculs effectués sur ces nombres font l'objet d'erreurs d'arrondi. Ces erreurs d'arrondi peuvent avoir des conséquences importantes et même, modifier le flot de contrôle du programme. Ainsi, la vérification de programmes qui effectuent des calculs sur les nombres flottants est un problème clef dans le processus de développement de systèmes critiques.

Les méthodes de vérification de programmes numériques sont principalement issues des méthodes classiques de vérification de programmes. La vérification bornée de modèles (BMC) a largement été utilisée pour détecter des bugs lors de la conception de processeurs [3] ou de logiciels [10]. Les solveurs SMT sont aujourd'hui présents dans la plupart des outils de BMC pour manipuler directement des formules de haut niveau [2, 8, 10]. L'outil CBMC de vérification bornée de modèles code chacune des opérations sur les flottants comme une fonction logique sur des vecteurs de bits. Cet encodage nécessite l'introduction de milliers de variables additionnelles et rend souvent le problème insoluble [6]. D'autres outils basés sur l'interprétation abstraite [9, 20] peuvent montrer l'absence d'erreurs d'exécution (e.g., la division par zéro) dans des programmes numériques. Grâce à une sur-

approximation du calcul sur les flottants, ces outils sont conservatifs de l'ensemble des solutions. Cependant, cette sur-approximation peut être très large et, par conséquent, ces outils rejettent un nombre significatif de programmes valides. La programmation par contraintes (CP) a aussi été utilisée pour la génération de cas de tests [12, 13] et la vérification de programmes [7]. La CP offre de multiples bénéfices tels que la capacité de déduire des informations de problèmes partiellement instanciés ou de fournir des contre-exemples. La programmation par contraintes est dotée d'un cadre très flexible qui facilite l'intégration de nouveau solveurs sur des domaines spécifiques comme les solveurs sur les flottants. Notons que les solveurs sur les réels ne sont pas capable de traiter correctement l'arithmétique sur les flottants. Des solveurs dédiés et corrects sont donc requis par les outils de programmation par contraintes et de vérification bornée de modèles pour tester ou vérifier des programmes numériques [1].

Les techniques existantes de résolution de contraintes sur les flottants sont basées sur une adaptation des consistances locales sur les réels (e.g. box-consistance, 2B-consistance) [19, 18, 5]. Toutefois, ces solveurs peinent à passer à l'échelle. C'est pourquoi nous introduisons ici une nouvelle méthode qui s'appuie sur des solveurs sur les réels pour filtrer des contraintes sur les nombres à virgule flottante. L'idée principale est de construire des relaxations fines et conservatives sur les réels pour des contraintes sur les flottants. Afin d'obtenir des relaxations fines, chaque opération sur les flottants est approximée selon le mode d'arrondi. Par exemple, supposons que x et y sont des nombres flottants positifs normalisés [2], alors le produit $x \otimes y$, pour un mode d'arrondi vers $-\infty$, sera borné par :

$$\alpha \times (x \times y) < x \otimes y \leq x \times y$$

où $\alpha = 1/(1 + 2^{-p+1})$ et p est la taille de la mantisse. Nous avons aussi défini des approximations fines pour les autres cas tels que l'addition avec un mode d'arrondi au plus près ou la multiplication par une constante.

À l'aide de ces relaxations, le problème initial sur les nombres flottants est tout d'abord transformé en un ensemble de contraintes non-linéaires sur les réels. Une linéarisation des contraintes non-linéaires est ensuite effectuée afin d'obtenir un problème linéaire mixte (MILP) sur les réels. Lors de ce processus, des variables binaires permettent de traiter les parties concaves tout en évitant l'utilisation de sur-approximations trop lâches. Ce dernier ensemble de contraintes peut directement être résolu par les solveurs MILP disponibles sur les réels qui, eux, ne sont par limités par l'arithmétique des nombres flottants. Les solveurs MILP efficaces utilisent des calculs en virgule flottante et peuvent donc perdre des solutions. Afin d'obtenir un comportement correct de tel solveurs, des procédures d'arrondi correct sont appliquées aux coefficients des relaxations [17, 4] et la méthode décrite dans [21] est utilisée pour calculer un minimum correct à partir du résultat incorrect fourni par un solveur MILP. Les résultats préliminaires sont très prometteurs et montrent que la technique introduite ici peut faciliter le passage à l'échelle des outils nécessitant un solveur de contraintes sur les flottants.

Notre méthode repose sur une représentation de haut niveau des opérations sur les flottants et ne souffre donc pas des mêmes limitations que celle reposant sur un encodage en vecteur de bits. Cet encodage utilisé par CBMC génère des milliers de variables binaires additionnelles pour chacune des opérations sur les flottants du programme. Par exemple, une addition de deux variables flottantes codées sur 32 bits demande 2554 variables binaires [6]. L'approximation mixte proposée dans [6] réduit notablement le nombre de ces variables. Mais le système résultant reste encore coûteux en mémoire. Notons qu'une simple addition avec seulement 5 bits de précision nécessite encore 1035 variables additionnelles. Notre méthode génère aussi des variables additionnelles : des variables intermédiaires sont ajouter afin de décomposer les expressions complexes en en opérations élémentaires sur les flottants et quelques variables binaires permettent de gérer les différents cas de nos relaxations. Cependant, la quantité de variables additionnelles ainsi générées reste négligeable en comparaison de celle requise par le modèle à base de vecteurs de bits.

1.1 Un exemple illustratif

Avant de détailler notre méthode, illustrons notre approche sur un exemple très simple. Considérons la contrainte suivante :

$$z = x + y - x \tag{1}$$

où x, y et z sont des variables de type flottant sur 32 bits. Sur les nombres réels, cette expression peut évidemment être réduite à $z = y$. Ce n'est pas le cas pour les nombres flottants. Par exemple, sur les

1. voir, par exemple, FPSE (http ://www.irisa.fr/celtique/carlier/fpse.html), un solveur de contraintes sur les flottants issue de programmes C.

2. Un nombre flottant x est déterminé par un triplet (s, e, m) où s est son signe, e son exposant et m sa mantisse. Sa valeur est donnée par $(-1)^s \times 1.m \times 2^e$. r et p spécifient le nombre de bits de son exposant et de sa mantisse. Dans la norme IEEE 754, les simples sont définis par $(r, p) = (8, 23)$ et les doubles par $(r, p) = (11, 52)$.

flottants et avec un mode d'arrondi au plus proche $10.0 + 10^{-8} - 10.0$ n'est pas égal à 10^{-8} mais à 0. Ce phénomène d'absorption montre bien pourquoi les expressions sur les flottants ne peuvent pas être simplifiées de la même manière que les expressions sur les réels.

Supposons que $x \in [0.0, 10.0]$, $y \in [0.0, 10.0]$ et $z \in [0.0, 10.0^8]$. FP2B, un algorithme de 2B-consistance [15] adapté aux flottants [5], propage les domaines de x et y vers le domaine de z en utilisant l'arithmétique des intervalles. La propagation inverse n'étant d'aucune utilité ici, le processus de filtrage donne :

$$x \in [0.0, 10.0],\ y \in [0.0, 10.0],\ z \in [0.0, 20.0]$$

Ce résultat souligne les difficultés des algorithmes de filtrages classiques à traiter les occurrences multiples. Une consistance plus forte comme la 3B-consistance [15] peut réduire le domaine de z à $[0.0, 10.01835250854492188]$. Par contre, la 3B-consistance ne réussit pas à réduire le domaine de z lorsque x et y ont tous les deux plus de deux occurrences comme dans la contrainte $z = x + y - x - y + x + y - x$.

L'algorithme introduit dans cet article construit d'abord une relaxation conservative non linéaire sur les réels du problème initial sur les flottants. Appliqué à la contrainte 1, il produit les relaxations sur les réels suivantes :

$$\begin{cases} (1 - \frac{2^{-p}}{(1-2^{-p})})(x+y) \leq tmp1 \\ tmp1 \leq (1 + \frac{2^{-p}}{(1+2^{-p})})(x+y) \\ (1 - \frac{2^{-p}}{(1-2^{-p})})(tmp1 - x) \leq tmp2 \\ tmp2 \leq (1 + \frac{2^{-p}}{(1+2^{-p})})(tmp1 - x) \\ z = tmp2 \end{cases}$$

où p est la taille de la mantisse de la variable flottante. $tmp1$ approxime le résultat de l'opération $x \oplus y$ à l'aide de deux plans sur les réels qui capturent les résultats de l'addition sur les flottants. $tmp2$ fait la même chose pour la soustraction. Certaines relaxations, comme la multiplication, contiennent des termes non-linéaires. Dans ce cas là, un processus de linéarisation des termes non-linéaires est appliqué afin d'obtenir un programme linéaire. Une fois le système complètement linéaire, un solveur MILP réduit le domaine de chaque variable en en calculant successivement le minimum et le maximum.

FPLP (Floating-point linear programming) est un outil qui implémente l'algorithme précédemment esquissé. Un appel à FPLP sur la contrainte (1) donne le résultat suivant :

$$x \in [0, 10],\ y \in [0, 10],\ z \in [0, 10.0000023841859]$$

qui est beaucoup plus précis que celui de FP2B. Contrairement à la 3B-consistance, FPLP donne le même résultat avec $z = x + y - x - y + x + y - x$.

1.2 Organisation de l'article

La suite de cet article est organisée comme suit : la section suivante introduit les relaxations non-linéaires sur les réels des contraintes sur les flottants. Le processus de linéarisation des relaxations est ensuite décrit avant de détailler l'algorithme de filtrage. Les résultats expérimentaux sont présentés dans la section qui précède la conclusion.

2 Relaxation des contraintes sur les flottants

Cette section introduit les relaxations sur les réels des contraintes sur les flottants issues du problème initial. Ces relaxations sont la pierre angulaire du processus de filtrage décrit dans cet article. Elles doivent non seulement être *correctes*, i.e., préserver l'ensemble des solutions du problème initial, mais aussi *fines*, i.e., inclure le moins possible de flottants non solution.

La construction de ces relaxations repose sur deux techniques : l'*erreur relative* et les opérations *correctement arrondies*. La première de ces techniques est communément utilisée pour analyser la précision du calcul. La seconde propriété est garantie par toute implémentation conforme au standard IEEE 754 de l'arithmétique des flottants : une opération correctement arrondie est une opération dont le résultat sur les flottants est égal à l'arrondi du résultat de l'opération équivalente sur les réels. En d'autre terme : soit x et y deux nombres flottants, $\odot$ et $\cdot$, respectivement, une opération sur les flottants et son équivalent sur les réels, si $\odot$ est correctement arrondie alors, $x \odot y = round(x \cdot y)$.

La suite de cette section détaille d'abord la construction de ces relaxations pour un cas particulier avant de la généraliser aux autres cas. Nous montrons ensuite comment les différents cas peuvent être simplifiés.

2.1 Un cas particulier

Afin d'expliquer comment obtenir ces relaxations, considérons d'abord le cas où le mode d'arrondi est vers $-\infty$ et le résultat de l'opération est un nombre flottant positif et normalisé. Un telle opération, dénotée $\odot$, peut être n'importe laquelle des quatre opérations de base de l'arithmétique des flottants. Toutes les opérandes sont supposées être du même type, i.e., float, double ou long double. On a alors la propriété suivante :

Proposition 1. *Soient x et y, des nombres flottants dont la mantisse possède p bits. Supposons que le mode d'arrondi soit fixé $-\infty$ et que le résultat de $x \odot y$ soit un nombre positif normalisé strictement inférieur à max_{float}, le plus grand des flottants, alors on a*

$$\frac{1}{1+2^{-p+1}}(x \cdot y) < x \odot y \leq (x \cdot y)$$

où $\odot$ une opération basique sur les nombres flottants et $\cdot$ est l'opération équivalente sur les nombres réels.

Démonstration. Selon la norme IEEE 754, les opérations sont correctement arrondies et le mode d'arrondi est fixé à $-\infty$. On a donc :

$$x \odot y \leq x \cdot y < (x \odot y)^+ < max_{float} \tag{2}$$

$(x \odot y)^+$, le successeur de $(x \odot y)$ dans l'ensemble des nombres à virgule flottante, peut être calculé par

$$(x \odot y)^+ = (x \odot y) + ulp(x \odot y)$$

puisque, par définition, $ulp(x) = x^+ - x$. Il résulte donc de (2) que

$$x \odot y \leq x \cdot y < (x \odot y) + ulp(x \odot y)$$

La seconde inéquation peut être réécrite comme suit :

$$\frac{1}{x \odot y + ulp(x \odot y)} < \frac{1}{x \cdot y}$$

En multipliant les deux parties de l'inéquation par $x \odot y$ – qui est un nombre positif – on obtient :

$$\frac{x \odot y}{x \odot y + ulp(x \odot y)} < \frac{x \odot y}{x \cdot y}$$

En multipliant chaque partie de l'inéquation précédente par -1 et en ajoutant 1 à chaque membre, on obtient

$$1 - \frac{x \odot y}{x \cdot y} < 1 - \frac{x \odot y}{x \odot y + ulp(x \odot y)} = \frac{ulp(x \odot y)}{x \odot y + ulp(x \odot y)} \tag{3}$$

Considérons maintenant ϵ, l'erreur relative définie par

$$\epsilon = \left| \frac{real\_value - float\_value}{real\_value} \right|$$

ϵ est la valeur absolue de la différence entre le résultat sur les réels et le résultat sur les flottants divisé par le résultat sur les réels. Dans notre cas, le résultat de $x \odot y$ étant un flottant positif normalisé, et sachant que $x \cdot y \geq x \odot y$, l'erreur relative est donnée par :

$$0 \leq \epsilon = \frac{x \cdot y - x \odot y}{x \cdot y} = 1 - \frac{x \odot y}{x \cdot y}$$

Donc, grâce à (3), on a

$$0 \leq \epsilon < \frac{ulp(x \odot y)}{x \odot y + ulp(x \odot y)}$$

z, le résultat de l'opération $x \odot y$, est un nombre flottant binaire positif et normalisé qui peut s'écrire $z = 1.m_z 2^{e_z}$, où m_z, sa mantisse, a p bits. Par ailleurs, $ulp(z) = 2^{-p+1}2^{e_z}$. On a donc,

$$0 \leq \epsilon < \frac{2^{-p+1}2^{e_z}}{m_z 2^{e_z} + 2^{-p+1}2^{e_z}} = \frac{2^{-p+1}}{m_z + 2^{-p+1}}$$

La valeur de la mantisse pour un nombre flottant normalisé appartient à l'intervalle $[1.0, 2.0[$. La borne supérieure de l'erreur relative ϵ est donnée par le minimum de $m_z + 2^{-p}$ qui est atteint lorsque $m_z = 1$. Donc

$$0 \leq \epsilon < \frac{2^{-p+1}}{1+2^{-p+1}}$$

Puisqu'on a

$$\epsilon = \frac{x \cdot y - x \odot y}{x \cdot y}$$

On a

$$0 \leq \frac{x \cdot y - x \odot y}{x \cdot y} < \frac{2^{-p+1}}{1+2^{-p+1}}$$

et

$$0 \leq x \cdot y - x \odot y < (x \cdot y)\frac{2^{-p+1}}{1+2^{-p+1}}$$

En multipliant chaque partie par -1 et en ajoutant $x \cdot y$, on obtient finalement

$$\frac{1}{1+2^{-p+1}}(x \cdot y) < x \odot y \leq x \cdot y$$

□

2.2 Généralisation

Le tableau 1 donne les relaxations pour les différents modes d'arrondi et les différent cas, i.e., les nombres flottants positifs ou négatifs, normalisés ou non. À chaque cas est associée une approximation correcte et fine obtenue de façon similaire à celle détaillée dans la section précédente.

Notez que certains cas particuliers permettent d'améliorer la précision des relaxations. Par exemple, l'addition avec un mode d'arrondi vers $\pm\infty$ peut être légèrement améliorée. La structure du problème offre aussi des possibilités d'amélioration des approximations. Par exemple, $2 \otimes x$ étant calculé exactement [3], cette expression peut directement être évaluée sur les réels.

3. S'il n'y a pas de dépassement de capacité.

Mode d' arrondi	Négatif normalisé	Négatif dénormalisé	Positif dénormalisé	Positif normalisé
vers $-\infty$	$[(1+2^{-p+1})z_r, z_r]$	$[z_r - min_f, z_r]$	$[z_r - min_f, z_r]$	$[\frac{1}{(1+2^{-p+1})}z_r, z_r]$
vers $+\infty$	$[z_r, \frac{1}{(1+2^{-p+1})}z_r]$	$[z_r, z_r + min_f]$	$[z_r, z_r + min_f]$	$[z_r, (1+2^{-p+1})z_r]$
vers 0	$\left[z_r, \frac{1}{(1+2^{-p+1})}z_r\right]$	$[z_r - min_f, z_r]$	$[z_r, z_r + min_f]$	$[\frac{1}{(1+2^{-p+1})}z_r, z_r]$
au plus près	$[(1+\frac{2^{-p}}{(1+2^{-p})})z_r,$ $(1-\frac{2^{-p}}{(1-2^{-p})})z_r]$	$[z_r - \frac{min_f}{2},$ $z_r + \frac{min_f}{2}]$	$[z_r - \frac{min_f}{2}$ $z_r + \frac{min_f}{2}]$	$[(1-\frac{2^{-p}}{(1-2^{-p})})z_r,$ $(1+\frac{2^{-p}}{(1+2^{-p})})z_r]$

TABLE 1 – Relaxations de $x \odot y$ pour chaque mode d'arrondi, avec $z_r = x \cdot y$.

2.3 Simplification des relaxations

Le principal problème des relaxations précédentes est que le processus de résolution doit traiter les différents cas. Ainsi, pour n opérations élémentaires, le solveur devra potentiellement traiter 4^n combinaisons de relaxations. Afin de diminuer substantiellement cette complexité, nous décrivons ici une combinaison des quatre cas liés à chaque arrondi en une unique relaxation.

Considérons d'abord le cas où le mode d'arrondi est fixé à $-\infty$:

Proposition 2. *Soient x et y deux nombres flottants dont la taille de la mantisse est p. Supposons que le mode d'arrondi est fixé $-\infty$ et que le résultat de $x \odot y$ est tel que $-max_{float} < x \odot y < max_{float}$. Alors, on a :*

$$z_r - 2^{-p+1}|z_r| - min_f \leq x \odot y \leq z_r$$

où min_f est le plus petit nombre flottant positif, $\odot$ et $\cdot$ sont, respectivement, une opération arithmétique de base sur les flottants et son équivalent sur les réels, et $z_r = x \cdot y$.

Démonstration. La première étape consiste à combiner les approximations des nombres normalisés et dénormalisés. Si $z_r > 0$ alors $\frac{1}{1+2^{-p+1}}z_r < z_r$. Donc,

$$\frac{1}{1+2^{-p+1}}z_r - min_f < z_r - min_f$$

et

$$\frac{1}{1+2^{-p+1}}z_r - min_f < \frac{1}{1+2^{-p+1}}z_r$$

Il s'en suit que :

$$\frac{1}{1+2^{-p+1}}z_r - min_f < x \odot y \leq z_r,\ z_r \geq 0$$

De la même manière, lorsque $z_r \leq 0$, on a :

$$(1+2^{-p+1})z_r - min_f < x \odot y \leq z_r,\ z_r \leq 0$$

Mode d'arrondi	Approximation de $x \odot y$
vers $-\infty$	$[z_r - 2^{-p+1}\|z_r\| - min_f, z_r]$
vers $+\infty$	$[z_r, z_r + 2^{-p+1}\|z_r\| + min_f]$
vers 0	$[z_r - 2^{-p+1}\|z_r\| - min_f,$ $z_r + 2^{-p+1}\|z_r\| + min_f]$
au plus près	$[z_r - \frac{2^{-p}}{(1-2^{-p})}\|z_r\| - \frac{min_f}{2},$ $z_r + \frac{2^{-p}}{(1-2^{-p})}\|z_r\| + \frac{min_f}{2}]$

TABLE 2 – Les relaxations simplifiées de $x \odot y$ pour chaque mode d'arrondi.(avec $z_r = x \cdot y$).

Ces deux approximations peuvent être réécrites comme suit :

$$\begin{cases} z_r - \frac{2^{-p+1}}{1+2^{-p+1}}z_r - min_f < x \odot y \leq z_r, & z_r \geq 0 \\ z_r + 2^{-p+1}z_r - min_f < x \odot y \leq z_r, & z_r \leq 0 \end{cases}$$

Pour combiner les approximations des cas positifs et négatifs, on utilise la valeur absolue :

$$\begin{cases} z_r - \frac{2^{-p+1}}{1+2^{-p+1}}|z_r| - min_f < x \odot y \leq z_r, & z_r \geq 0 \\ z_r - 2^{-p+1}|z_r| - min_f < x \odot y \leq z_r, & z_r \leq 0 \end{cases}$$

Puisque $max\{\frac{2^{-p+1}}{1+2^{-p+1}}, 2^{-p+1}\} = 2^{-p+1}$, on a

$$z_r - 2^{-p+1}|z_r| - min_f \leq x \odot y \leq z_r$$

□

Le même raisonnement s'applique aux autres cas. Le tableau 2 donne les relaxations simplifiées pour chaque mode d'arrondi. Notez que ces approximations définissent un espace concave.

3 Linéarisation des relaxations

Les relaxations introduites dans les sections précédentes contiennent des termes non-linéaires qui ne peuvent pas être directement manipulés par un solveur MILP. Cette section décrit comment ces termes sont approximés par un ensemble de contraintes linéaires.

3.1 Linéarisation de la valeur absolue

Les relaxations simplifiées précédentes contiennent des valeurs absolues. Elles peuvent être soit grossièrement approximées par trois inégalités ou implémentées grâçe à une décomposition classique plus fine basée sur une réécriture utilisant des "big M" :

$$\begin{cases} z = z_p - z_n \\ |z| = z_p + z_n \\ 0 \leq z_p \leq M \times b \\ 0 \leq z_n \leq M \times (1-b) \end{cases}$$

où b est une variable booléenne, z_p et z_n sont des variables réelles positives, et M est un grand nombre flottant tel que $M \geq max\{|\underline{z}|, |\overline{z}|\}$. La méthode sépare z_p, les valeurs positives de z, de z_n, ses valeurs négatives. Lorsque $b = 1$, z prend ses valeurs positives avec $z = z_p = |z|$. Si $b = 0$, z prend ses valeurs négatives avec $z = -z_n$ et $|z| = z_n$.

Lorsque le solveur MILP permet l'utilisation de contraintes d'indicateur, les deux dernières contraintes peuvent alors être remplacé par :

$$\begin{cases} b = 0 \rightarrow z_p = 0 \\ b = 1 \rightarrow z_n = 0 \end{cases}$$

3.2 Linéarisation des opérations non-linéaires

La linéarisation du produit, du carrée, et de la division sont basées sur les techniques standard proposées par Sahinidis et al [22]. Elles ont aussi été utilisées dans la Quad [14] pour résoudre des contraintes sur les réels. $x \times y$ est linéarisé selon Mc Cormick [16] comme suit :

Supposons que $x \in [\underline{x}, \overline{x}]$ et $y \in [\underline{y}, \overline{y}]$, alors

$$\begin{cases} L1 : z - \underline{x}y - \underline{y}x + \underline{x}\underline{y} \geq 0 \\ L2 : -z + \underline{x}y + \overline{y}x - \underline{x}\overline{y} \geq 0 \\ L3 : -z + \overline{x}y + \underline{y}x - \overline{x}\underline{y} \geq 0 \\ L4 : z - \overline{x}y - \overline{y}x + \overline{x}\overline{y} \geq 0 \end{cases}$$

Ces linéarisations ont été prouvées optimales par Al-Khayyal et Falk [1].

Lorsque $x = y$, i.e., dans le cas où $z = x \otimes x$, la linéarisation peut être améliorée : l'espace convexe de x^2 est sous-estimé par l'ensemble des tangentes à la courbe de x^2 entre $\underline{x}$ et $\overline{x}$, et sur-estimé par la ligne qui joint $(\underline{x}, \underline{x}^2)$ à $(\overline{x}, \overline{x}^2)$. Un bon compromis est obtenu en ne prenant que les deux tangentes correspondant aux bornes de x. La linéarisation de x^2 donne donc :

$$\begin{cases} L1 : z + \underline{x}^2 - 2\underline{x}x \geq 0 \\ L2 : z + \overline{x}^2 - 2\overline{x}x \geq 0 \\ L3 : (\underline{x} + \overline{x})x - z - \underline{x}\overline{x} \geq 0 \\ L4 : z \geq 0 \end{cases}$$

La division peut bénéficier des propriétés de l'arithmétique sur les réels : observons simplement que $z = x/y$ est équivalent à $x = z \times y$. Les linéarisations de Mc Cormick [16] s'appliquent donc aussi à ce cas. Elles nécessitent cependant les bornes de z qui sont calculables à l'aide de l'arithmétique par intervalles :

$$[\underline{z}, \overline{z}] = [\nabla(min(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y})), \Delta(max(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y}))]$$

où ∇ et Δ sont respectivement les modes d'arrondis vers $-\infty$ et $+\infty$.

4 Algorithme de filtrage

L'algorithme de filtrage proposé s'appuie sur les linéarisations des relaxations sur les réels du problème initial sur les flottants pour tenter de réduire les domaines des variables au moyen d'un solveur MILP. L'algorithme 1 détaille les étapes de ce processus de filtrage.

Initialement, les contraintes sur les flottants sont relaxées vers des contraintes non-linéaires sur les réels. Puis, les termes non-linéaires de ces relaxations sont linéarisés afin d'obtenir un programme linéaire mixte.

Algorithm 1 FPLP

1: **Function** $FPLP(\mathcal{V}, \mathcal{D}, \mathcal{C}, \epsilon)$
2: % $\mathcal{V}$: Variables flottantes
3: % $\mathcal{D}$: Domaines des variables
4: % $\mathcal{C}$: Contraintes sur les flottants
5: % ϵ : Réduction minimale entre deux itérations
6: $\mathcal{C}' \leftarrow$ **Approximate** $(\mathcal{C})$;
7: $\mathcal{C}'' \leftarrow$ **Linearise** $(\mathcal{C}', \mathcal{D})$;
8: $reduction \leftarrow 0$;
9: **repeat**
10: $\quad \mathcal{D}' \leftarrow$ **FP2B**$(\mathcal{V}, \mathcal{D}, \mathcal{C}, \epsilon)$;
11: $\quad \mathcal{C}'' \leftarrow$ **UpdateLinearisations**$(\mathcal{C}'', \mathcal{D}')$;
12: $\quad oldReduction \leftarrow reduction$;
13: $\quad$ **for all** $x \in \mathcal{V}$ **do**
14: $\quad\quad [\underline{x}_{\mathcal{D}'}, \overline{x}_{\mathcal{D}'}] \leftarrow [$**safeMin**$(x, \mathcal{C}''),$ $-$**safeMin**$(-x, \mathcal{C}'')]$;
15: $\quad$ **end for**
16: $\quad reduction \leftarrow \sum_{x \in \mathcal{V}} ((\overline{x}_{\mathcal{D}} - \underline{x}_{\mathcal{D}}) - (\overline{x}_{\mathcal{D}'} - \underline{x}_{\mathcal{D}'}))$;
17: $\quad \mathcal{D} \leftarrow \mathcal{D}'$
18: **until** $reduction \leq \epsilon \times oldReduction$;
19: **return** $\mathcal{D}$;

Le processus de filtrage commence par un appel à FP2B, un filtrage qui s'appuie sur une adaptation de la 2B-consistance aux contraintes sur les flottants, qui fait une première tentative de réduction des bornes des variables. FP2B permet aussi la propagation des

bornes des variables aux variables intermédiaires. Le coût de ce filtrage est relative léger (w est fixé à 10%). La réduction des bornes opérées par FP2B facilite le travail du solveur MILP.

Après cela, le solveur MILP est utilisé afin de calculer la borne supérieure et la borne inférieure du domaine de chacune des variables.

Ce processus est répété jusqu'à ce que le pourcentage de réduction des domaines des variables soit inférieur à un ϵ donné.

4.1 Correction du solveur MILP

L'utilisation d'un solveur efficace MILP tel que CPLEX pour filtrer les domaines des variables soulève deux problèmes importants liés à l'usage de calculs sur les flottants.

Premièrement, les coefficients des linéarisations sont calculés en utilisant l'arithmétique des nombres à virgule flottante et sont donc sujet à des erreurs d'arrondi. Par conséquent, afin d'éviter la perte de solutions, ces calculs doivent reposer sur des directions d'arrondi correctement choisies. Par exemple, considérons la linéarisation de x^2 avec $\overline{x} \geq \underline{x} \geq 0$, on a alors

$$\begin{cases} L1 : y + \Delta(\underline{x}^2) - \Delta(2\underline{x})x \geq 0 \\ L2 : y + \Delta(\overline{x}^2) - \Delta(2\overline{x})x \geq 0 \\ L3 : \Delta(\underline{x} + \overline{x})x - y - \nabla(\underline{x}\overline{x}) \geq 0 \\ L4 : y \geq 0 \end{cases}$$

Ces directions correctes d'arrondi nous assure que l'ensemble des solutions est préservé. Le calcul de coefficients corrects est plus largement détaillé dans [17, 4].

Deuxièmement, les solveurs MILP efficaces utilisent l'arithmétique sur les nombres flottants. Donc, le minimum qu'ils calculent peut être erroné. Le solveur MILP incorrect est rendu correct grâce à la procédure de correction introduite dans [21]. Elle consiste en un calcul d'une borne inférieure correcte du minimum global.

5 Expérimentations

Cette section compare les résultats de différentes techniques de filtrage de contraintes sur les nombres flottants avec la méthode introduite dans cet article. Les expérimentations ont été faites sur un PC portable équipé d'un processeur Intel Duo Core 2.8Ghz, de 4Go de mémoire et dans un environnement Linux.

Nos expérimentations sont basées sur l'ensemble de des programmes suivants :

- `Absorb 1` détecte si x absorbe y lors d'une simple addition ; `Absorb 2` vérifie si y absorbe x.
- `Fluctuat1` est un chemin dans un programme extrait d'un article qui présente l'outil Fluctuat [11]. `Fluctuat2` est un autre chemin du même programme `Fluctuat`.
- `MeanValue` est un programme qui retourne vrai si un intervalle contient une solution en utilisant le théorème des accroissement finis.
- `Cosine` est un programme qui calcule la fonction $cos()$ avec une formule de Taylor.
- `SqrtV1` calcule la racine carrée d'un nombre dans l'intervalle $[0.5, 2.5]$ en utilisant une méthode itérative à deux variables.
- `SqrtV2` calcule la racine carrée en utilisant les séries de Taylor.

La tableau 3 présente les résultats des expérimentations faites avec les méthodes de filtrage suivantes : FP2B, une adaptation de l'algorithme de la 2B-consistance aux contraintes sur les flottants, FP3B, une adaptation de l'algorithme de la 3B-consistance aux contraintes sur les flottants, FPLP(without 2B), une implémentation de l'algorithme 1 sans appel intermédiaire à FP2B, et, FPLP, une implémentation de l'algorithme 1. La première colonne du tableau 3 donne le nom du programme, la deuxième, le nombre de variables du problème initial, et la troisième, le nombre de variables intermédiaires utilisées pour décomposer les contraintes initiales complexes en opérations élémentaires. La quatrième colonne donne le nombre de variables binaires utilisées par FPLP. Pour chaque algorithme de filtrage, le tableau 3 donne le temps nécessaire au filtrage des contraintes (colonnes $t(ms)$). Pour tout les algorithmes de filtrage – à l'exception de FP2B – le tableau 3 donne le pourcentage de réduction obtenu par rapport à celle obtenue en utilisant la FP2B (colonne %(FP2B)). Notez qu'une limite de 2 minutes pour terminer le filtrage est imposée dans ces expérimentations.

Les résultat obtenus dans le tableau 3 montrent que FPLP effectue de meilleures réductions des domaines en moins de temps que les méthodes de filtrage locales. Ce phénomène est particulièrement accentué sur les exemples `Absorb1` et `SqrtV1`. Dans ces deux exemples, FP2B souffre du problème des occurrences multiples de variables. FPLP présente de manière consistante de meilleurs performances que FP3B : il fourni presque toujours des domaines plus petits et nécessite toujours moins de temps.

Une comparaison entre FPLP sans et avec appel à FP2B montre l'intérêt d'une coopération entre ces deux méthodes de filtrage qui peut réduire considérablement le temps de calcul sans modifier les capacités de filtrage.

				FP2B	FP3B		FPLP (without 2B)		FPLP	
Programme	n	n_T	n_B	$t(ms)$	$t(ms)$	%(FP2B)	$t(ms)$	%(FP2B)	$t(ms)$	%(FP2B)
`Absorb1`	2	1	1	TO	TO	-	3	98.91	5	98.91
`Absorb2`	2	1	1	1	24	0.00	3	100.00	4	100.00
`Fluctuat1`	3	12	2	4	156	99.00	264	99.00	172	99.00
`Fluctuat2`	3	10	2	1	4	0.00	29	0.00	21	0.00
`MeanValue`	4	28	6	3	82	97.45	530	97.46	78	97.46
`Cosine`	5	33	7	5	153	33.60	104	33.61	43	33.61
`SqrtV1`	11	140	29	9	27198	99.63	1924	100.00	1187	100.00
`SqrtV2`	21	80	17	7	TO	-	6081	100.00	1321	100.00

TABLE 3 – Expérimentations

6 Conclusion

Dans cet article, nous avons introduit un nouvel algorithme de filtrage pour les contraintes sur les nombres flottants. Grâce aux linéarisations des relaxations non-linéaires sur les réels des contraintes initiales sur les flottants, cet algorithme peut réduire les domaines des variables avec un solveur MILP. Les expérimentations montrent que FPLP améliore significativement le processus de filtrage, en particulier, lorsqu'il est associé à un algorithme de filtrage de type FP2B. La programme linéaire mixte bénéficie d'une vue plus globale du problème que les filtrage locaux et peut ainsi apporter une solution au problème des occurrences multiples de variables.

Plus d'expérimentations sont cependant nécessaire afin d'améliorer notre compréhension des interactions entre les deux algorithmes ainsi que leur performances. Par exemple, il n'est pas encore certain qu'un appel systématique à FPLP à chaque nœud de l'arbre de recherche soit requis.

Références

[1] F.A. Al-Khayyal and J.E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, pages 8 :2 :273–286, 1983.

[2] Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania. Bounded model checking of software using smt solvers instead of sat solvers. *Int. J. Softw. Tools Technol. Transf.*, 11 :69–83, January 2009.

[3] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, 1999. Springer-Verlag.

[4] Glencora Borradaile and Pascal Van Hentenryck. Safe and tight linear estimators for global optimization. *Mathematical Programming*, 2005.

[5] Bernard Botella, Arnaud Gotlieb, and Claude Michel. Symbolic execution of floating-point computations. *Softw. Test., Verif. Reliab.*, 16(2) :97–121, 2006.

[6] Angelo Brillout, Daniel Kroening, and Thomas Wahl. Mixed abstractions for floating-point arithmetic. In *Proceedings of FMCAD 2009*, pages 69–76. IEEE, 2009.

[7] Hélène Collavizza, Michel Rueher, and Pascal Hentenryck. CPBPV : a constraint-programming framework for bounded program verification. *Constraints*, 15(2) :238–264, 2010.

[8] Lucas Cordeiro, Bernd Fischer, and Joao Marques-Silva. Smt-based bounded model checking for embedded ansi-c software. *IEEE Transactions on Software Engineering*, May 2011.

[9] Patrick Cousot, Radhia Cousot, Jerome Feret, Antoine Mine, Laurent Mauborgne, David Monniaux, and Xavier Rival. Varieties of static analyzers : A comparison with astree. In *TASE '07*, pages 3–20. IEEE, 2007.

[10] Malay K Ganai and Aarti Gupta. Accelerating high-level bounded model checking. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, ICCAD '06, pages 794–801, New York, NY, USA, 2006. ACM.

[11] K. Ghorbal, E. Goubault, and S. Putot. A logical product approach to zonotope intersection. In *Computer Aided Verification*, pages 212–226. Springer, 2010.

[12] Arnaud Gotlieb, Bernard Botella, and Michel Rueher. Automatic test data generation using constraint solving techniques. In *ISSTA*, pages 53–62, 1998.

[13] Arnaud Gotlieb, Bernard Botella, and Michel Rueher. A clp framework for computing structural test data. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Computational Logic*, volume 1861 of *Lecture Notes in Computer Science*, pages 399–413. Springer, 2000.

[14] Yahia Lebbah, Claude Michel, Michel Rueher, David Daney, and Jean-Pierre Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM J. Numer. Anal*, 42 :2076–2097, 2005.

[15] Olivier Lhomme. Consistency techniques for numeric csps. In *IJCAI*, pages 232–238, 1993.

[16] G.P. McCormick. Computability of global solutions to factorable nonconvex programs – part i – convex underestimating problems. *Mathematical Programming*, 10 :147–175, 1976.

[17] C. Michel, Y. Lebbah, and M. Rueher. Safe embedding of the simplex algorithm in a CSP framework. In *Proc. of CPAIOR 2003, CRT, Université de Montréal*, pages 210–220, 2003.

[18] Claude Michel. Exact projection functions for floating point number constraints (pdf). In *AMAI*, 2002.

[19] Claude Michel, Michel Rueher, and Yahia Lebbah. Solving constraints over floating-point numbers. In Toby Walsh, editor, *CP*, volume 2239 of *Lecture Notes in Computer Science*, pages 524–538. Springer, 2001.

[20] Antoine Miné. *Weakly Relational Numerical Abstract Domains*. PhD thesis, École Polytechnique, Palaiseau, France, December 2004.

[21] A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer programming. *Math. Programming A.*, page 99 :283 296, 2004.

[22] Hong S. Ryoo and Nikolaos V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, pages 107–138, 1996.

Clustering sous contraintes utilisant SAT

J.-P. Métivier P. Boizumault B. Crémilleux M. Khiari S. Loudni

Université de Caen – GREYC – (CNRS UMR 6072)
Campus II – Côte de Nacre – Boulevard du Maréchal Juin
14000 Caen – France
prenom.nom@unicaen.fr

Résumé

Le clustering a pour objectif de partitionner un ensemble de données (transactions) en groupes (clusters) d'une manière telle que les transactions appartenant à un même cluster soient similaires mais différentes de celles appartenant aux autres clusters. Le clustering sous contraintes a pour objectif de trouver des clusters plus pertinents en spécifiant, sous forme de contraintes, les propriétés requises. Nous avons proposé [16] un langage à base de contraintes pour l'extraction de motifs combinant plusieurs motifs locaux. Dans cet article, nous nous intéressons à la mise en œuvre en SAT de notre langage de contraintes. Pour cela, nous présentons le codage réalisé et nous montrons comment il tire parti des caractéristiques des solveurs SAT. Les expérimentations effectuées avec `MiniSat` montrent la faisabilité et l'intérêt de notre approche.

Abstract

Clustering aims at partitioning data into groups (clusters) so that transactions occurring in the same cluster are similar but different from those appearing in other clusters. Clustering is an important and popular data mining task and, by nature, clustering proceeds by iteratively refining queries until a satisfactory solution is found. In this application paper, we first present a constraint based language for modeling such queries. The usefulness of our approach is highlighted by several examples coming from the clustering based on associations. Then we show how each constraint (and each query) can be encoded in SAT and solved taking benefit from several features of SAT solvers. Experiments performed using `MiniSat` show the feasibility and the interest of our approach.

1 Introduction

Le clustering est une activité importante en fouille de données dont l'objectif est de partitionner un ensemble de données (transactions) en groupes (clusters) d'une manière telle que les transactions appartenant à un même cluster soient similaires mais différentes de celles appartenant aux autres clusters [12].

Le clustering sous contraintes [3] a pour objectif de trouver des clusters plus pertinents en spécifiant, sous forme de contraintes, les propriétés requises. Parmi les contraintes les plus communément utilisées figurent les contraintes imposant que des transactions appartiennent (ou non) à un même cluster [20], que les clusters aient une taille minimale et/ou maximale [2], qu'une transaction appartienne à un cluster donné, ...

Dans un travail précédent [16], nous avons proposé un langage à base de contraintes pour l'extraction de motifs combinant plusieurs motifs locaux. L'utilisateur modélise son problème sous forme d'une requête initiale (conjonction de contraintes), puis procède par raffinements successifs de requêtes jusqu'à ce qu'une solution satisfaisante soit trouvée. Le langage de contraintes proposé porte sur des termes construits à partir de constantes, de variables, d'opérateurs et de symboles de fonctions. Cette approche tire bénéfice des travaux récents sur la fertilisation croisée entre la fouille de données et la programmation par contraintes [14, 17].

Dans cet article, nous nous intéressons à la mise en œuvre en SAT de notre langage de contraintes. Pour cela, nous présentons l'encodage réalisé et nous montrons comment il tire parti des caractéristiques des solveurs SAT : clauses binaires, propagation unitaire, réseaux de tri, ...Les expérimentations effectuées avec `MiniSat` sur différents jeux de données de l'UCI montrent la faisabilité et l'intérêt de notre approche.

La section 2 rappelle très brièvement les caractéristiques de notre langage de contraintes qui sont utilisées dans cet article. Les apports de notre approche par raf-

finements successifs sont illustrés au travers de deux types de clustering : le clustering conceptuel (cf section 3.2) et le clustering exploitant les connaissances a priori du domaine (cf section 3.3). La section 4 présente l'encodage réalisé. La section 5 montre la faisabilité et l'intérêt de notre approche aux travers de différentes expérimentations menées avec `MiniSat`. La section 6 dresse un bref état de l'art des rares travaux menés dans ce domaine.

2 Aperçu succinct de notre langage de contraintes

Soit $\mathcal{I}$ un ensemble de n littéraux distincts appelés *items*. Un motif ensembliste d'items (ou *itemset*) est un sous-ensemble non vide de $\mathcal{I}$. Ces motifs forment le langage $\mathcal{L}_\mathcal{I} = 2^\mathcal{I} \backslash \emptyset$. Un *contexte transactionnel* $\mathcal{T}$ est défini comme un multi-ensemble de m motifs de $\mathcal{L}_\mathcal{I}$. Chacun de ces motifs, appelé transaction, constitue une entrée de la base de données.

Les contraintes sont des relations portant sur des termes, ces termes étant construits en utilisant des constantes, des variables, des opérateurs et des symboles de fonction. L'utilisateur peut définir de nouveaux symboles de fonctions aussi bien que de nouvelles contraintes.

Les termes sont construits à partir de :

- de constantes qui peuvent être des valeurs numériques, des motifs ou des transactions,
- de variables, notées X_j, pour $1 \leq j \leq k$, représentant les motifs inconnus,
- d'opérateurs ensemblistes (e.g., $\cap, \cup, \setminus$) ou numériques (e.g., $+, -, \times, /$),
- de symboles de fonction portant sur un ou plusieurs motifs tels que :
  - $\texttt{cover}(X_j) = \{t \mid t \in \mathcal{T}, X_j \subseteq t\}$ est le support de X_j, i.e. l'ensemble des transactions couvertes par X_j.
  - $\texttt{freq}(X_j) = \mid \{t \mid t \in \mathcal{T}, X_j \subseteq t\} \mid$ est la fréquence du motif X_j.
  - $\texttt{size}(X_j) = \mid \{i \mid i \in \mathcal{I}, i \in X_j\} \mid$ est la cardinalité du motif X_j.
  - $\texttt{overlapItems}(X_i, X_j) = \mid X_i \cap X_j \mid$ est le nombre d'items communs à X_i et X_j.
  - $\texttt{overlapTransactions}(X_i, X_j) = \mid \texttt{cover}(X_i) \cap \texttt{cover}(X_j) \mid$ est le nombre de transactions couvertes à la fois par X_i et X_j.

Les contraintes sont des relations portant sur les termes. On distingue trois types de contraintes prédéfinies : les contraintes numériques (e.g., $<, \leq, =, \neq, \geq, >$), les contraintes ensemblistes (e.g., $=, \neq, \in, \notin, \subset, \subseteq$), et les contraintes spécifiques telles que :

- $\texttt{isEmpty}(X_j)$ est satisfaite ssi $X_j \neq \emptyset$
- $\texttt{closed}(X_j)$ est satisfaite ssi X_j est un motif fermé [1].
- $\texttt{coverTransactions}([X_1, ..., X_k])$ est satisfaite ssi chaque transaction est couverte par au moins un motif ($\bigcup_{1 \leq i \leq k} \texttt{cover}(X_i) = \mathcal{T}$)
- $\texttt{noOverlapTransactions}([X_1, ..., X_k])$ est satisfaite ssi $\forall i, j$ t.q. $1 \leq i < j \leq k$, $\texttt{cover}(X_i) \cap \texttt{cover}(X_j) = \emptyset$
- $\texttt{coverItems}([X_1, ..., X_k])$ est satisfaite ssi chaque item appartient à au moins un motif ($\cup_{1 \leq i \leq k} X_i = \mathcal{I}$)
- $\texttt{noOverlapItems}([X_1, ..., X_k])$ est satisfaite ssi $\forall i, j$ t.q. $1 \leq i < j \leq k$, $X_i \cap X_j = \emptyset$
- $\texttt{canonical}([X_1, ..., X_k])$ est satisfaite ssi $\forall i$ t.q. $1 \leq i < k$, le motif X_i est inférieur au motif X_{i+1} par rapport à l'ordre lexicographique.

Enfin, une requête est une conjonction de contraintes. Pour un aperçu complet du langage, se reporter à [16].

3 Clustering par raffinements successifs

Un atout majeur de notre approche est de fournir à l'utilisateur un cadre flexible et efficace pour modéliser et raffiner ses requêtes. En pratique, l'utilisateur commence par écrire une requête initiale Q_1 qu'il peut raffiner par des opérations successives d'ajout/retrait de contraintes (Q_{i+1} est dérivée à partir de Q_i) jusqu'à ce l'information extraite soit considérée comme suffisamemt relevante. Nous illustrons cette approche par raffinements successifs grâce à deux exemples de clustering à base d'associations : le clustering conceptuel (cf section 3.2) et le clustering exploitant les connaissances du domaine (cf section 3.3).

3.1 Modélisation d'une requête de clustering

Un problème de clustering peut être décrit par un scénario dans lequel l'utilisateur cherche à obtenir une partition $\pi_T = (X_1, ..., X_k)$ d'une base de données $\mathcal{T}$, contenant m points, en k clusters (non vides). Notre langage de contraintes permet de définir $\texttt{isPartition}([X_1,...,X_k])$ vérifiant que les clusters $(X_1,...,X_k)$ constituent une partition de $\mathcal{T}$:

$$\texttt{isPartition}([X_1, ..., X_k]) \equiv \left\{ \begin{array}{l} \wedge_{1 \leq i \leq k} \texttt{notEmpty}(X_i) \wedge \\ \texttt{coverInstances}([X_1, ..., X_k]) \wedge \\ \texttt{noOverlapInstances}([X_1, ..., X_k]) \end{array} \right.$$

Un problème de clustering contient naturellement des solutions symétriques. Soit $s = (\pi_1, ..., \pi_k)$ une solution contenant k clusters π_i. Toute permutation de

1. soit $Tr_j = \texttt{cover}(X_j)$. X_j est fermé ssi X_j est le plus grand ($\subset$) motif couvrant Tr_j.

ces k motifs est aussi une solution. La contrainte `canonical`$([X_1, ..., X_k])$ est alors utilisée pour éviter les solutions symétriques. Elle impose que pour tout i t.q. $1 \leq i < k$, le cluster X_i est inférieur au cluster X_{i+1} par rapport à l'ordre lexicographique. Nous pouvons ainsi définir `isClustering`$([X_1, ..., X_k])$ qui évite les solutions symétriques :

$$\texttt{isClustering}([X_1, ..., X_k]) \equiv \left\{ \begin{array}{l} \texttt{isPartition}([X_1, ..., X_k]) \wedge \\ \texttt{canonical}([X_1, ..., X_k]) \end{array} \right.$$

3.2 Clustering conceptuel

Soit $\mathcal{I}$ un ensemble de n items et $\mathcal{T}$ un ensemble de m transactions. Les motifs fermés sont de bons candidats pour la recherche de clusterings à base d'associations. En effet, ils rassemblent un maximum de similitude entre un ensemble de transactions. Un problème de clustering standard peut être formulé comme suit : trouver un ensemble de k motifs fermés $X_1, X_2, ..., X_k$ (i.e., clusters) couvrant toutes les transactions sans chevauchement des supports respectifs de ces motifs (requête Q_1).

$$\texttt{isConceptualCl}([X_1, ..., X_k]) \equiv \left\{ \begin{array}{l} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i \leq k}\ \texttt{closed}(X_i) \end{array} \right.$$

Eviter les clusters peu fréquents. Quand un cluster extrait est de faible fréquence, il est considéré comme peu représentatif et donc peu fiable. Une contrainte de fréquence minimale (par rapport à un seuil δ_1) peut alors être ajoutée.

$$\left\{ \begin{array}{l} \texttt{isConceptualCl}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i \leq k}\ \texttt{freq}(X_i) \geq \delta_1 \end{array} \right.$$

Eviter les clusters de petite taille. Un clustering contenant au moins un motif X_i de petite taille [2] n'est pas considéré comme pertinent parce que X_i ne garantit pas suffisamment de similarité entre les transactions associées. Une contrainte de cardinalité minimale (par rapport à un seuil δ_2) peut alors être ajoutée (requête Q_2).

$$\left\{ \begin{array}{l} \texttt{isConceptualCl}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i \leq k}\ \texttt{freq}(X_i) \geq \delta_1 \wedge \\ \wedge_{1 \leq i \leq k}\ \texttt{size}(X_i) \geq \delta_2 \end{array} \right.$$

Equilibrage des clusters. Dans les problèmes de clustering conceptuel, on a généralement tendance à préférer les solutions dans lesquelles les fréquences des différents clusters extraits sont proches les unes des autres. Pour chaque couple de clusters (X_i, X_j), leur différence de fréquence doit être inférieure à un seuil $\Delta \times m$ où Δ est un pourcentage (requête Q_3).

$$\left\{ \begin{array}{l} \texttt{isConceptualCl}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq k}\ \mid \texttt{freq}(X_i) - \texttt{freq}(X_j) \mid \leq \Delta \times m \end{array} \right.$$

D'une manière analogue, il est possible de modéliser d'autres problèmes de clustering [5] tels que le soft clustering, le co-clustering, ou encore le soft co-clustering.

Soft clustering est une version relaxée du problème de clustering dans lequel on autorise un certain chevauchement entre les supports des motifs extraits (inférieur à un seuil δ_T).

$$\left\{ \begin{array}{l} \wedge_{1 \leq i \leq k}\ \texttt{closed}(X_i) \wedge \\ \texttt{coverTransactions}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq k}\ \texttt{overlapTransactions}(X_i, X_j) \leq \delta_T \wedge \\ \texttt{canonical}([X_1, ..., X_k]) \end{array} \right.$$

Co-clustering vise à trouver k clusters couvrant à la fois l'ensemble des transaction et l'ensemble des items, sans chevauchement ni entre les motifs ni entre leurs supports.

$$\left\{ \begin{array}{l} \texttt{isConceptualCl}([X_1, ..., X_k]) \wedge \\ \texttt{coverItems}([X_1, ..., X_k]) \wedge \\ \texttt{noOverlapItems}([X_1, ..., X_k]) \end{array} \right.$$

Soft co-clustering est une version relaxée du co-clustering dans lequel on autorise un certain chevauchement entre les motifs extraits (inférieur à un seuil δ_I) et un certain chevauchement entre leurs supports (inférieur à un seuil δ_T).

$$\left\{ \begin{array}{l} \wedge_{1 \leq i \leq k}\ \texttt{closed}(X_i) \wedge \\ \texttt{coverTransactions}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq k}\ \texttt{overlapTransactions}(X_i, X_j) \leq \delta_T \wedge \\ \texttt{coverItems}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq k}\ \texttt{overlapItems}(X_i, X_j) \leq \delta_I \wedge \\ \texttt{canonical}([X_1, ..., X_k]) \end{array} \right.$$

3.3 Clustering exploitant les connaissances a priori

Lors d'applications réelles, l'utilisateur possède souvent des connaissances a priori (portant sur le domaine ou la base de données) qui peuvent être utiles pour faire émerger des groupes de données. Certaines méthodes de clustering ne savent pas tirer parti de ces connaissances [3, 5]. De telles connaissances sont souvent exprimées par des contraintes portant sur les transactions (e.g., les contraintes `mustLink` et `cannotLink` [19, 20]), et des contraintes portant sur les clusters (e.g., les contraintes de *Diamètre Maximal* et de *Séparation Minimale* [7, 8]).

Dans cette section, nous décrivons comment ces contraintes peuvent être modélisées en utilisant notre langage de contraintes. Puis, nous montrons comment

2. Les motifs de taille 1 par exemple, reflètent souvent des valeurs qui ne sont apparues que lors de la binarisation d'un attribut.

elles peuvent être combinées pour obtenir des requêtes plus pertinentes imposant des tailles minimales de clusters ou cherchant des clusterings équilibrés. Soit $\mathcal{T}$ une base de données composée de m transactions. Soit $d(t_1, t_2)$ une distance définie sur les transactions.

Contraintes au niveau des transactions :
- $\texttt{mustLink}(t_1, t_2)$ impose que les transactions t_1 et t_2 appartiennent au même cluster.
- $\texttt{cannotLink}(t_1, t_2)$ impose que les transactions t_1 et t_2 n'appartiennent pas au même cluster.

Contrainte de diamètre maximal. Le *diamètre d'un cluster* X_j est la distance maximale entre toute paire de transactions appartenant au support de X_j. La contrainte de diamètre maximal impose que le diamètre de chaque cluster soit au plus égal à une valeur donnée α. Pour toute paire de transactions (t_i, t_j) t.q. $d(t_i, t_j) > \alpha$, t_i et t_j ne pourront appartenir à un même cluster et devront satisfaire la contrainte $\texttt{cannotLink}(t_i, t_j)$.

Contrainte de séparation minimale. La *séparation entre deux clusters* X_i *et* X_j est la distance minimale entre toute paire de transactions, l'une appartenant au support de X_i et l'autre au support de X_j. La contrainte de séparation minimale impose que la séparation entre deux clusters soit au moins égale à une valeur donnée β. Pour toute paire de transactions (t_i, t_j) t.q. $d(t_i, t_j) < \beta$, t_i et t_j devront appartenir au même cluster et satisfaire la contrainte $\texttt{mustLink}(t_i, t_j)$.
Ces différents types de contraintes peuvent être combinés (requête Q'_1).

$$\begin{cases} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) < \beta} \texttt{mustLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) > \alpha} \texttt{cannotLink}(t_i, t_j) \end{cases}$$

Des connaissance a priori sur des transactions et/ou clusters donnés peuvent aisément être modélisées. L'appartenance (resp. la non appartenance) de deux transactions t_{i_0} et t_{j_0} données au même cluster est modélisée par l'ajout de la contrainte : $\texttt{mustLink}(t_{i_0}, t_{j_0})$ (resp. $\texttt{cannotLink}(t_{i_0}, t_{j_0})$). L'appartenance (resp. la non appartenance) d'une transaction t_{i_0} donnée au cluster X_{j_0} est modélisée par l'ajout de la contrainte : $t_{i_0} \in X_{j_0}$ (resp. $t_{i_0} \notin X_{j_0}$).

Affiner les requêtes. D'une manière analogue au clustering conceptuel, les clusters de petite taille sont considérés comme non-pertinents et peuvent être ignorés (requête Q'_2).

$$\begin{cases} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) < \beta} \texttt{mustLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) > \alpha} \texttt{cannotLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i \leq k} \texttt{size}(X_i) \geq \delta \wedge \end{cases}$$

Et rechercher des clusterings équilibrés (requête Q'_3).

$$\begin{cases} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) < \beta} \texttt{mustLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) > \alpha} \texttt{cannotLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq k} \mid \texttt{size}(X_i) - \texttt{size}(X_j) \mid \leq \Delta \times m \end{cases}$$

4 Encodage sous la forme d'une CNF

Après avoir sélectionné le jeu de données et avoir modélisé la requête, il est nécessaire de traduire l'ensemble sous forme d'une CNF qui sera composée :
- de la CNF associée à la recherche des différents motifs inconnus (cf sections 4.1 et 4.2) ;
- des CNFs associées à l'encodage de chaque contrainte figurant dans la requête (cf sections 4.3 à 4.5).

Le clustering conceptuel, où chaque cluster est représenté par un motif fermé, requiert un encodage plus volumineux que le clustering avec connaissances a priori (cf sections 4.1 et 4.2). L'encodage des contraintes de partitionnement est le même dans les deux cas (cf section 4.3). Les contraintes de seuil sont encodées grâce à des réseaux de tri, qui permettent un encodage indépendant de la valeur du seuil (cf section 4.5). Finalement, nous montrons comment les propriétés d'inférence des contraintes `mustLink` et `cannotLink` sont automatiquement déduites par le solveur SAT (cf section 4.6) et comment ce dernier peut être est rendu complet grâce à l'utilisation de *restarts* (cf section 4.7).

4.1 Encodage du clustering conceptuel

Soit $\mathcal{I}$ un ensemble de n items et $\mathcal{T}$ un ensemble de m transactions. Notons $(d_{t,i})$ la matrice booléenne de taille (m, n) telle que $(d_{t,i}$=`True`$)$ ssi $(i \in t)$. Les motifs inconnus sont modélisés grâce à des variables booléennes et à cette matrice $(d_{t,i})$.

Soit $X_1, X_2, ..., X_k$ un ensemble de k motifs inconnus. De la même façon que [14, 17], le lien entre le jeu de données et les motifs inconnus X_j est établi à l'aide de deux ensembles de variables :
- $X_{1,j}, X_{2,j}, ..., X_{n,j}$ t.q. $(X_{i,j}$=`True`$)$ ssi $(i \in X_j)$
- $T_{1,j}, T_{2,j}, ..., T_{m,j}$ t.q. $(T_{t,j}$=`True`$)$ ssi $(X_j \subset t)$

Considérons le motif inconnu X_j, la relation de couverture $(X_j \subset t)$ peut alors être modélisée par la CNF suivante :

$$\bigwedge_{\{i \in \mathcal{I} \mid \neg d_{t,i}\}} \neg X_{i,j} \tag{1}$$

La relation entre X_j et $\mathcal{T}$ est modélisée en établissant que, pour chaque transaction t, $(T_{t,j}$=`True`$)$ ssi X_j couvre t :

$$\forall t \in \mathcal{T}, T_{t,j} \Leftrightarrow (X_j \subset t) \tag{2}$$

En utilisant Eq. 1, l'implication (gauche-droite) de Eq. 2 peut être modélisée par la CNF suivante :

$$\bigwedge_{t\in\mathcal{T}}(\bigwedge_{\{i\in\mathcal{I}|\neg d_{t,i}\}}(\neg T_{t,j}\vee\neg X_{i,j})) \quad (3)$$

En utilisant Eq. 1, l'implication (droite-gauche) de Eq. 2 peut être modélisée par la CNF suivante :

$$\bigwedge_{t\in\mathcal{T}}((\bigvee_{\{i\in\mathcal{I}|d_{t,i}\}}X_{i,j})\vee T_{t,j}) \quad (4)$$

Enfin, Eq. 3 et Eq. 4 doivent être vérifiées pour tout motif X_j. L'encodage nécessite donc $(k\times m\times n)$ clauses binaires.
La contrainte $\texttt{closed}(X_j)$ est modélisée [3] par la formule suivante :

$$\bigwedge_{i\in\mathcal{I}}(X_{i,j}\Leftrightarrow\bigwedge_{\{t\in\mathcal{T}|\neg d_{t,i}\}}\neg T_{t,j})$$

4.2 Encodage du clustering avec connaissances a priori

Soit $\mathcal{T}$ un ensemble de m transactions. Chaque cluster inconnu est modélisé à l'aide de m variables booléennes $T_{t,j}$ telles que $(T_{t,j}=True)$ ssi la transaction t appartient au cluster j. Un cluster est désigné ici par son index (compris entre 1 et k) et non plus par un motif fermé. C'est pourquoi, l'encodage de ce type de clustering est de plus petite taille que celui du clustering conceptuel.

4.3 Encodage des contraintes de partitionnement

L'encodage de ces contraintes est le même pour les deux types de clustering car il n'utilise que les variables $T_{t,j}$.

- $\texttt{coverTransactions}([X_1,...,X_k])$ impose que chaque transaction $t\in\mathcal{T}$ appartienne à au moins un cluster. Ainsi, pour chaque $t\in\mathcal{T}$, il existe au moins un cluster X_j t.q. t figure dans X_j.

$$\bigwedge_{t\in\mathcal{T}}(\bigvee_{j\in[1..k]}T_{t,j})$$

- $\texttt{noOverlapTransactions}([X_1,...,X_k])$ s'assure que chaque transaction $t\in\mathcal{T}$ appartient à au plus un cluster. Une transaction t appartient à au moins deux clusters ssi il existe X_i et X_j t.q. $(t\in X_i)\wedge(t\in X_j)$, c'est-à-dire $\bigvee_{1\leq i<j\leq k}(T_{t,i}\wedge T_{t,j})$. Il ne reste plus qu'à prendre la négation de cette formule pour chaque $t\in\mathcal{T}$.

$$\bigwedge_{t\in\mathcal{T}}(\bigwedge_{1\leq i<j\leq k}(\neg T_{t,i}\vee\neg T_{t,j}))$$

- $\texttt{notEmpty}(X_j)$ impose qu'il existe au moins une transaction $t\in\mathcal{T}$ appartenant au cluster X_j et est modélisée par la clause :

$$\bigvee_{t\in\mathcal{T}}T_{t,j}$$

- $\texttt{canonical}([X_1,...,X_k])$ impose que pour tout i t.q. $1\leq i<k$, X_i soit inférieur à X_{i+1} et est encodée grâce à un comparateur binaire.

4.4 Encodage des contraintes mustLink et cannotLink

$\texttt{mustLink}(t_1,t_2)$ contraint deux transactions t_1 et t_2 à appartenir à un même cluster. Ainsi, pour chaque $j\in[1..k]$, $T_{t_1,j}\Leftrightarrow T_{t_2,j}$.

$$\bigwedge_{1\leq j\leq k}(\neg T_{t_1,j}\vee T_{t_2,j})\wedge(T_{t_1,j}\vee\neg T_{t_2,j})$$

$\texttt{cannotLink}(t_1,t_2)$ contraint deux transactions t_1 et t_2 à ne pas appartenir au même cluster. Ainsi, pour chaque $j\in[1..k]$, $T_{t_1,j}\Leftrightarrow\neg T_{t_2,j}$.

$$\bigwedge_{1\leq j\leq k}(\neg T_{t_1,j}\vee\neg T_{t_2,j})\wedge(T_{t_1,j}\vee T_{t_2,j})$$

Notons que ces contraintes sont encodées en utilisant $(2\times k)$ clauses binaires.

4.5 Encodage des contraintes de seuil à l'aide de réseaux de tri

Les contraintes de seuil sont directement modélisées à l'aide de contraintes de cardinalité portant sur les variables booléennes définies aux sections 4.1 et 4.2. Par exemple, $\texttt{freq}(X_j)\geq\ \delta_1$ se modélise $\#(T_{1,j},T_{2,j},...,T_{m,j})\geq\delta_1$, et $\texttt{size}(X_j)\geq\delta_2$ se modélise $\#(X_{1,j},X_{2,j},...,X_{n,j})\geq\delta_2$. Il en est de même pour les autres contraintes de seuil.

Plusieurs encodages performants des contraintes de cardinalité ont été proposés [1, 18]. Ces travaux utilisent des additionneurs unaires permettant d'établir la cohérence d'arc via la propagation unitaire. Mais de tels encodages requièrent une nombre quadratique de clauses [1] ou dépendent de la valeur du seuil [18] (plus grand est le seuil, plus grande est la CNF résultante). Dans le cas du clustering, ces seuils sont souvent très grands, et la taille de l'encodage devient rapidement prohibitive. C'est pourquoi nous avons choisi d'utiliser les réseaux de tri pour encoder les contraintes de cardinalité. Tout d'abord, la taille de cet encodage ne dépend plus de la valeur de seuil. De plus, l'utilisation de réseaux de tri permet aussi d'établir la cohérence d'arc via la propagation unitaire [11].

3. Soit Tr_j=$\texttt{cover}(X_j)$. X_j est un motif fermé (i.e. le plus grand motif couvrant Tr_j) ssi $\forall i\in\mathcal{I}, i\in X_j\Leftrightarrow\forall t\in\mathcal{T}, i\notin t\Rightarrow X_j\not\subset t$. Cette modélisation repose sur les propriétés de la connexion de Galois (voir [17] pour plus de détails).

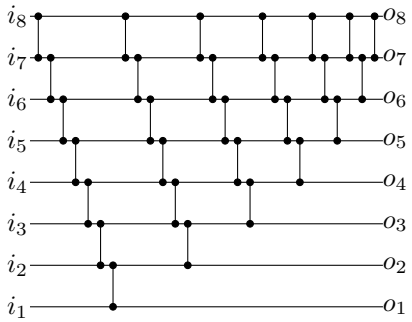


FIGURE 1 – Réseau de tri bulles (n=8).

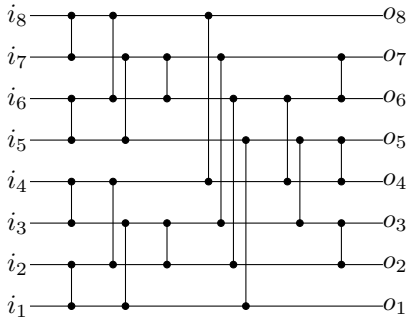


FIGURE 2 – Réseau de tri *Batcher* (n=8).

Les réseaux de tri peuvent être utilisés pour compter le nombre de littéraux positifs. Pour n entrées, le réseau associe n sorties qui correspondent à une version triée des entrées. Cette sortie triée constitue une représentation unaire du nombre de littéraux positifs. Les réseaux de tri utilisent une unique opération d'échange binaire et l'appliquent plusieurs fois afin de trier complètement les n littéraux en entrée. Un échange binaire prend en entrée deux littéraux i_1 et i_2, et retourne deux littéraux o_1 et o_2 tel que o_1 est le minimum entre i_1 i_2 ($o_1 \Leftrightarrow i_1 \wedge i_2$) et o_2 leur maximum ($o_2 \Leftrightarrow i_1 \vee i_2$).

Plusieurs algorithmes de tri ont été implémentés à l'aide des réseaux de tri. La figure 1 représente le réseau associé au tri bulles pour n=8 entrées. Chaque ligne horizontale représente un littéral tout au long du processus de tri, et chaque segment vertical représente un échange binaire entre les deux littéraux à ses extrémités.

Les réseaux de tri fournissent seulement une représentation unaire du nombre de littéraux positifs. Pour modéliser une contrainte de seuil, il est nécessaire de contraindre les littéraux en sortie du réseau. Ainsi, pour contraindre au moins δ littéraux à être positifs, le littéral o_δ doit être instancié à `True`. De la même façon, pour contraindre au plus δ littéraux à être positifs, le littéral $o_{\delta+1}$ doit être instancié à `False`.

L'utilisation d'un tri naïf, comme le tri bulles, peut conduire à un grand nombre d'échanges et donc à un encodage trop volumineux ($O(n^2)$). Nous avons choisi d'utiliser le tri *odd-even Batcher sort* [4] qui requiert moins d'échanges ($O(n \times (log\ n)^2)$). La figure 2 représente le réseau de tri associé à cet algorithme de tri pour n=8. Ce réseau de tri a déjà été utilisé dans `MiniSat+` et s'est avéré très efficace par rapport aux autres encodages des contraintes de cardinalité [11].

4.6 Transitivité des `mustLink` et `cannotLink`

Soit G=(V, E) un graphe constitué de contraintes `mustLink` où V=$\mathcal{T}$ et il existe une arête entre t_i et t_j ssi il existe une contrainte `mustLink`(t_i, t_j) [3, 20]. Soit CC_1 et CC_2 deux composantes connexes de G. *(i) s'il existe une contrainte* `mustLink`(t_1, t_2) *t.q.* $t_1 \in CC_1$ *et* $t_2 \in CC_2$*, alors il est possible d'inférer les contraintes* `mustLink`(x, y) *pour tout* $x \in CC_1$ *et tout* $y \in CC_2$*.* À l'inverse de `mustLink`, `cannotLink` n'est pas une relation d'équivalence, mais *(ii) s'il existe une contrainte* `cannotLink`(t_1, t_2) *t.q.* $t_1 \in CC_1$ *et* $t_2 \in CC_2$*, alors il est possible d'inférer les contraintes* `cannotLink`(x, y) *pour tout* $x \in CC_1$ *et tout* $y \in CC_2$*.*

De telles déductions sont habituellement faites en ajoutant toutes les contraintes `mustLink` et `cannotLink` ainsi inférées [3, 20]. Mais, grâce à l'utilisation de notre encodage (cf. section 4.4), il n'est plus nécessaire d'effectuer ces ajouts. En effet, toutes les contraintes déduites seront implicitement prises en compte par le solveur SAT.

4.7 Établir la complétude

Pour une CNF donnée, un solveur SAT, soit retourne une instanciation des variables (et seulement une) telle que la CNF est évaluée à `True`, soit prouve qu'il n'existe aucune instanciation la satisfaisant. Afin d'assurer la complétude de notre approche, plusieurs *restarts* sont réalisés. La CNF $\mathcal{F}$ modélisant la requête soumise est transmise au solveur. Après avoir obtenue une solution s_i, la négation de cette solution $\neg s_i$ est ajoutée à $\mathcal{F}$. Ce processus est itéré tant que le solveur SAT trouve de nouvelles solutions. L'utilisation de *restarts* peut sembler naïve, mais est en pratique très efficace (cf section 5). Cela est notamment dû au fait que $\mathcal{F}$ contienne un très grand nombre de clauses binaires qui permettent un filtrage par propagation unitaire très efficace.

5 Expérimentations

Nous avons utilisé le solveur `MiniSat` [4] [10] pour effectuer différentes expérimentations sur plusieurs jeux de données de l'UCI [5] ainsi qu'un jeu de données réelles

4. `http://minisat.se/`
5. `http://www.ics.uci.edu/~mlearn/MLRepository.html`

Jeu de données	m	n	Densité
`Australian`	653	125	0.40
`Mushroom`	8124	119	0.19
`P.-Tumor`	336	36	0.48
`Soybean`	630	50	0.32
`Zoo`	101	36	0.44
`Meningitis`	329	82	0.26

TABLE 1 – Caractéristiques des jeux de données.

nommé `Meningitis` et provenant de l'Hôpital Central de Grenoble. `Meningitis` recense les pathologies des enfants atteints d'une méningite virale ou bactérienne. La table 1 résume les différentes caractéristiques de ces jeux de données. La densité d'un jeu de données est $d=\sum d_{t,i}/(n \times m)$ (cf section 4.1).

Le processeur utilisé est un Core2Duo E8400 (2.83GHz) avec 4Go de RAM. Pour chaque expérimentation, nous avons reporté les temps CPU nécessaires au calcul de la première et des dix premières solutions en fonction du nombre k de clusters souhaités dans la requête. Le symbole `'-'` indique l'absence de solutions pour une requête.

5.1 Clustering conceptuel

La figure 3 indique les temps CPU nécessaires au calcul de la première et des dix premières solutions pour la requête Q_1 (cf section 3.2), ceci pour différentes valeurs de k. Notre approche est efficace, même pour des valeurs de k relativement grandes, et des jeux de données possédant un très grand nombre de motifs fermés (`Australian` possède plus de 20 millions de motifs fermés).

La figure 4 indique les temps CPU pour la requête Q_2 avec $\delta_1=m/10$ et $\delta_2=2$ (cf section 3.2). Les temps CPU augmentent car Q_2 est plus contrainte que Q_1 et comporte des contraintes de seuil. `Mushroom` est le jeu de données de plus grande taille. Il requiert une très grande quantité de clauses pour encoder les contraintes de seuil (1.4 million par motif inconnu [6]). Pour k=10, il n'y a de solution pour aucun jeu de données. En effet, il est impossible de trouver un clustering constitué de 10 motifs fermés différents ayant une fréquence supérieure à $m/10$ puisqu'aucun recouvrement n'est autorisé entre les m transactions.

La figure 5 indique les temps CPU nécessaires au calcul de la première solution pour la recherche d'un clustering équilibré (requête Q_3). Le ratio Δ est fixé à 20% (cf partie gauche de la figure 5) et à 40% (cf partie droite de la figure 5). Les temps CPU augmentent mais demeurent raisonnables.

La figure 6 décrit l'évolution du temps CPU en fonction du nombre de solutions pour le jeu de données `P.-Tumor` avec k=6. Une courbe est associée à chacune des trois requêtes Q_1 (rouge), Q_2 (bleu) et Q_3 avec Δ=40% (noir). Il est à noter que la requête Q_3 possède seulement 18 solutions. Les temps CPU pour Q_2 et Q_3 sont plus élevés que pour Q_1 mais demeurent abordables pour ce jeu de données. La figure 7 décrit la même évolution pour `Australian` avec k=6. Ce jeu de données est le pire compte-tenu du très grand nombre de motifs fermés qu'il contient.

La figure 8 indique la taille des encodages [7] des requêtes Q_1, Q_2 et Q_3 pour les jeux de données `Australian` et `Mushroom`. Même si l'encodage peut être de grande taille (plusieurs millions de clauses), le ratio r de clauses binaires demeure toujours très élevé.

5.2 Clustering exploitant les connaissances a priori

Ces expérimentations ont été menées sur les mêmes jeux de données que pour le clustering conceptuel (cf table 1) en utilisant la distance de Hamming entre deux transactions [8]. Le diamètre maximal α a été fixé à $n/20$ et la séparation minimale β à $n/2$.

La figure 9 indique les temps CPU nécessaires au calcul de la première et des dix premières solutions pour la requête Q'_1 (cf section 3.3), ceci pour différentes valeurs de k. Pour chaque jeu de données, les dix premières solutions (si elles existent) sont obtenues très rapidement, y compris pour `Mushroom` qui est le jeu de données de plus grande taille.

La figure 10 indique les temps CPU nécessaires au calcul de la première solution pour la recherche d'un clustering équilibré (requête Q'_3 de la section 3.3). Le ratio Δ a été fixé à 10% (cf partie gauche de la figure 10) et à 20% (cf partie droite de la figure 10). Même avec des contraintes additionnelles de seuil, les temps CPU demeurent raisonnables. Enfin, ces seuils étant élevés, peu de requêtes possèdent (au moins) une solution.

La figure 11 décrit l'évolution du temps CPU en fonction du nombre de solutions pour le jeu de données `Zoo` avec k=6. Une courbe est associée à chacune des trois requêtes Q'_1 (rouge), Q'_2 avec $\delta=m/10$ (bleu) et Q'_3 avec Δ=20% (noir). Ces courbes semblent être quasi-linéaires. Chacune des trois requêtes comporte de nombreuses contraintes `mustLink` et `cannotLink` qui sont encodées à l'aide de clauses binaires. Dès qu'une transaction est affectée à un cluster, de nombreuses déductions vont être effectuées grâce à la propagation unitaire et à la transitivité de ces contraintes (cf section 4.6). Mais, cette justification se doit d'être vérifiée de manière plus précise par de nouvelles expérimentations. La figure 12 donne des résultats similaires pour le jeu de données `Australian` (k=6).

6. C'est 50 fois moins que l'encodage proposé par Bailleux et 10 fois moins que celui proposé par Sinz (cf section 4.5).

7. Une unité représente 1 millier de litéraux ou de clauses.

8. Toute autre distance aurait pu être utilisée car celle-ci ne sert qu'à imposer les contraintes `mustLink` et `cannotLink`.

jeu de données	k=2	k=4	k=6	k=8	k=10
`Australian`	0.03	0.06	0.41	12.95	132.3
`Mushroom`	0.47	3.86	10.95	58.32	25.58
`P.-Tumor`	0.01	0.02	0.05	0.05	0.08
`Soybean`	0.01	0.07	1.05	1.04	1.09
`Zoo`	0.01	0.01	0.02	0.02	0.06
`Meningitis`	0.02	0.04	0.28	3.30	9.76

k=2	k=4	k=6	k=8	k=10
0.04	0.69	1.67	27.21	190.2
-	14.17	31.65	123.9	134.7
0.01	0.03	0.06	0.22	0.79
-	0.19	2.36	7.04	4.36
0.01	0.01	0.02	0.03	0.08
0.03	0.22	5.28	7.26	20.19

FIGURE 3 – (Q_1) Temps en s. pour la 1ère sol. (gauche) et les 10 premières sol. (droite).

jeu de données	k=2	k=4	k=6	k=8	k=10
`Australian`	0.18	1.35	305.8	2233	-
`Mushroom`	3.39	58.73	2715	-	-
`P.-Tumor`	-	0.09	4.55	104.3	-
`Soybean`	-	1.27	72.27	395.1	-
`Zoo`	0.02	0.02	0.10	0.51	-
`Meningitis`	-	4.14	186.5	1523	-

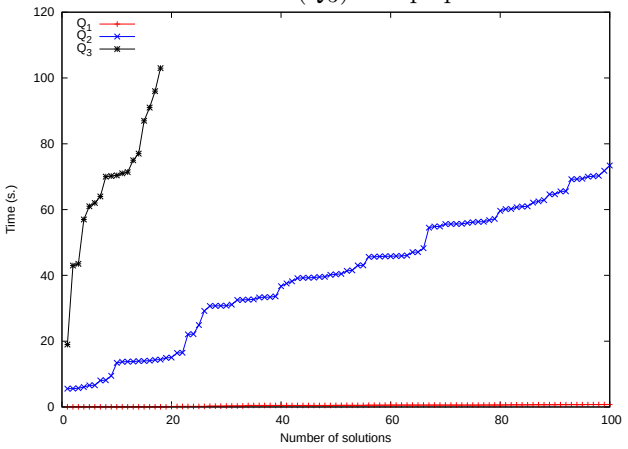
k=2	k=4	k=6	k=8	k=10
0.40	10.38	464.1	21856	-
-	114.9	10125	-	-
-	0.64	11.66	234.6	-
-	7.76	120.5	824.0	-
-	0.06	0.28	2.61	-
-	29.44	664.8	5748	-

FIGURE 4 – (Q_2) Temps en s. pour la 1ère sol. (gauche) et les 10 premières sol. (droite).

jeu de données	k=2	k=4	k=6	k=8	k=10
`Australian`	0.14	31.1	909.5	13274	-
`Mushroom`	-	-	-	-	-
`P.-Tumor`	0.04	0.94	-	66.45	900.7
`Soybean`	-	-	-	-	-
`Zoo`	-	-	-	3.59	1.31
`Meningitis`	-	-	-	-	-

k=2	k=4	k=6	k=8	k=10
0.12	4.78	407.0	5285	-
4.11	-	-	-	35645
0.04	0.44	18.95	108.5	91.9
0.11	2.37	-	-	1043
0.02	0.11	0.48	0.60	0.52
-	-	-	3730	-

FIGURE 5 – (Q_3) Temps pour la 1ère sol. avec Δ=20% (gauche) et Δ=40% (droite).



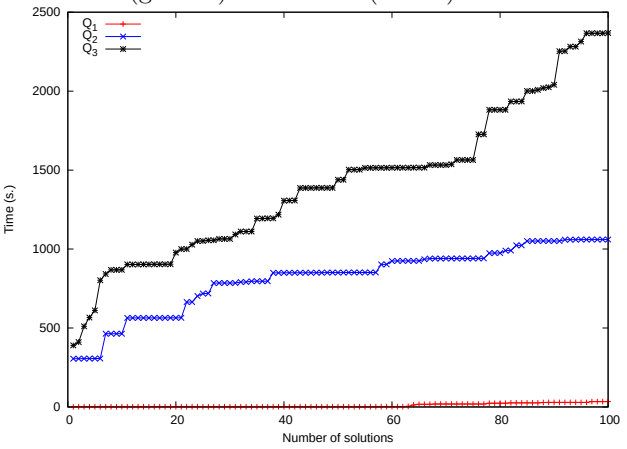


FIGURE 6 – Temps pour `P.-Tumor` (k=6).

`Australian`		$k(=2)$	$(k=6)$	$(k=10)$
Q_1	#littéraux	2.4	6.5	10.6
	#clauses	102.6	315.2	538.3
	ratio %	97.1	97.1	97.2
Q_2	#littéraux	104.6	312.9	521.3
	#clauses	409.1	1235	2071
	ratio %	74.3	74.5	74.6
Q_3	#littéraux	98.7	295.3	491.9
	#clauses	392.6	1185	1988
	ratio %	74.7	74.9	75.0

FIGURE 7 – Temps pour `Australian` (k=6).

`Mushroom`		$(k=2)$	$(k=6)$	$(k=10)$
Q_1	#littéraux	24.8	58.7	92.6
	#clauses	1650	5018	8516
	ratio %	98.5	98.8	98.9
Q_2	#littéraux	1341	4008	6675
	#clauses	5600	16868	28298
	ratio %	76.1	76.2	76.3
Q_3	#littéraux	1335	3990	6646
	#clauses	5598	16863	28291
	ratio %	76.1	76.3	76.5

FIGURE 8 – Nombre de littéraux et de clauses (une unité = 1000).

jeu de données	k=4	k=8	k=12	k=16	k=20
`Australian`	-	0.02	0.25	0.89	1.59
`Mushroom`	0.96	1.38	3.94	6.64	-
`P.-Tumor`	-	0.03	0.08	0.12	0.13
`Soybean`	0.01	0.03	0.12	0.16	-
`Zoo`	0.01	0.01	0.02	0.03	0.03

k=4	k=8	k=12	k=16	k=20
-	0.05	0.31	0.96	1.70
1.28	1.53	5.38	8.24	-
-	0.03	0.11	0.15	0.17
0.02	0.05	0.14	0.18	-
0.01	0.01	0.03	0.04	0.04

FIGURE 9 – (Q'_1) Temps en s. pour la 1ère sol. (gauche) et les 10 premières sol. (droite).

jeu de données	k=4	k=8	k=12	k=16	k=20
`Australian`	-	19.7	27.46	45.23	180.5
`Mushroom`	16.9	-	-	-	-
`P.-Tumor`	-	1.06	2.22	6.16	6.05
`Soybean`	-	-	-	-	-
`Zoo`	0.01	0.09	-	-	-

k=4	k=8	k=12	k=16	k=20
-	7.20	13.10	46.15	69.76
19.6	-	-	-	-
-	0.91	3.30	3.88	6.08
-	-	-	-	-
0.03	0.10	-	-	-

FIGURE 10 – (Q'_3) Temps pour la 1ère sol. avec Δ=10% (gauche) et Δ=20% (droite).

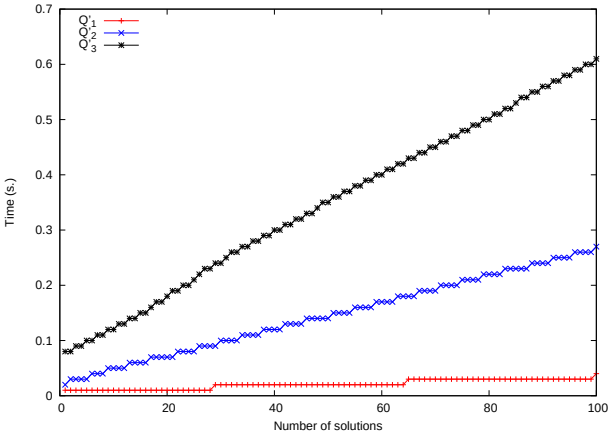


FIGURE 11 – Temps pour `Zoo` (k=6).

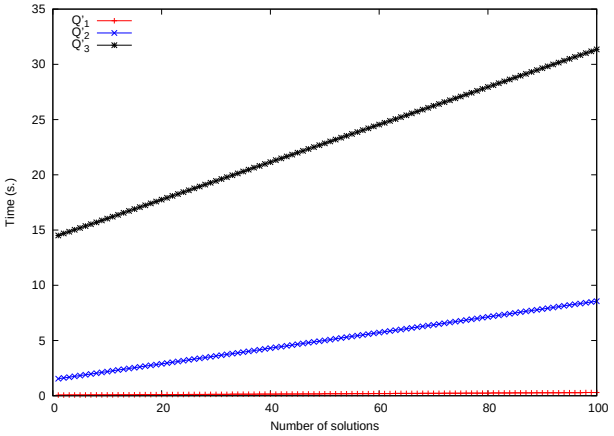


FIGURE 12 – Temps pour `Australian` (k=6).

`Australian`		($k=2$)	($k=6$)	($k=10$)
Q'_1	#littéraux	2.6	10.4	18.2
	#clauses	29.8	107.8	196.3
	ratio %	86.9	81.2	83.93
Q'_2	#littéraux	98.8	299.2	499.5
	#clauses	318.7	974.5	1640
	ratio %	68.6	68.3	68.5
Q'_3	#littéraux	98.8	299.2	499.5
	#clauses	318.7	974.5	1640
	ratio %	68.6	68.3	68.5

`Mushroom`		($k=2$)	($k=6$)	($k=10$)
Q'_1	#littéraux	32.4	129.9	227.4
	#clauses	1141	3651	6291
	ratio %	95.7	93.1	92.9
Q'_2	#littéraux	1343	4062	6781
	#clauses	5075	15452	25960
	ratio %	73.2	72.9	73.1
Q'_3	#littéraux	1343	4062	6781
	#clauses	5075	15452	25960
	ratio %	73.2	72.9	73.1

FIGURE 13 – Nombre de littéraux et de clauses (une unité = 1000).

La figure 13 indique la taille des encodages des requêtes Q'_1, Q'_2 et Q'_3 pour les jeux de données `Australian` et `Mushroom`. La taille de la CNF associée dépend grandement des valeurs α et β. Pour nos expérimentations, le diamètre maximal α a été fixé à $n/20$ et la séparation minimale β à $n/2$. La figure 13 indique aussi le ratio r de clauses binaires dans la CNF. Comme pour le clustering conceptuel (cf figure 8), l'encodage d'une requête peut être volumineux (plusieurs millions de clauses), mais le ratio r de clauses binaires demeure toujours élevé. Enfin, il est à noter que les tailles des CNFs associées à Q'_2 et Q'_3 sont très proches. En effet, seuls les seuils des contraintes de cardinalité sont différents (cf section 4.5).

Pour conclure, ces expérimentations montrent que les solveurs SAT permettent de résoudre de manière efficace ces deux types de clustering. Ils peuvent trouver, dans des temps très compétitifs, les premières solutions (ou prouver qu'il n'en existe pas), même pour des jeux de données difficiles (comme `Australian`) ou de grande taille (comme `Mushroom`).

6 Travaux relatifs

SAT pour le clustering. L'introduction des contraintes sur les transactions (`mustLink` et `cannotLink`) et des contraintes sur les clusters (Diamètre Maximal et Séparation Minimale) dans l'algorithme de clustering `k-means` [15] est décrite dans [7]. Une étude de complexité de ces deux familles de contraintes est présentée dans [8]. Davidson&al ont proposé la première approche utilisant SAT pour le clustering [9], mais elle ne traite que le cas très particulier où k=2, qui conduit

à une modélisation qui est une instance de 2-SAT. Gilpin&Davidson se sont intéressés au clustering hiérarchique [13] et montrent comment des *dendograms* peuvent être modélisés comme des instances du problème Horn-SAT.
SAT pour l'extraction de motifs. Une approche SAT pour résoudre le problème EMS *(Enumerating all Motifs in a Sequence)* a été proposée dans [6] et appliquée à deux problèmes concrets en bio-informatique et en sécurité informatique.

7 Conclusion et perspectives

Nous avons présenté l'encodage SAT de différents problèmes de clustering et nous avons montré comment il tire parti des caractéristiques des solveurs SAT : clauses binaires, propagation unitaire, réseaux de tri, ...Les expérimentations effectuées avec `MiniSat` sur différents jeux de données de l'UCI montrent la faisabilité et l'intérêt de notre approche.

Plusieurs améliorations peuvent être apportées à cet encodage pour en accélérer les performances : heuristiques de sélection de variables, exploitation des *backdoors* et des *nogoods*, ... Le passage à l'échelle, pour des valeurs de k plus grandes, sera aussi traité. Enfin, nous étudierons comment intégrer des critères d'optimisation dans cette approche.

Références

[1] O. Bailleux and Y. Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *CP'03*, volume 2833 of *LNCS*, pages 108–122. Springer, 2003.

[2] A. Banerjee and J. Ghosh. Scalable clustering algorithms with balancing constraints. *Data Min. Knowl. Discov.*, 13(3) :365–395, 2006.

[3] S. Basu, I. Davidson, and Kiri L. Wagstaff. *Constrained Clustering : Advances in Algorithms, Theory, and Applications*. Chapman & Hall, 2008.

[4] K. E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. Thomson Book Company, Washington D.C., 1968.

[5] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, USA, 2002.

[6] E. Coquery, S. Jabbour, and L. Sais. A constraint programming approach for enumerating motifs in a sequence. In *Workshop on Declarative Pattern Mining, ICDM 2011*, pages 1091–1097, Vancouver, Canada, December 2011.

[7] I. Davidson and S. Ravi. Clustering with constraints : Feasibility issues and the k-means algorithm. In *SDM*, 2005.

[8] I. Davidson and S. Ravi. Using instance-level constraints in agglomerative hierarchical clustering : theoretical and empirical results. *Data Min. Knowl. Discov.*, 18(2) :257–282, 2009.

[9] I. Davidson, S. Ravi, and L. Shamis. A SAT-based framework for efficient constrained clustering. In *SDM*, pages 94–105. SIAM, 2010.

[10] N. Eén and N. Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518, 2003.

[11] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *JSAT*, 2(1-4) :1–26, 2006.

[12] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2) :139–172, 1987.

[13] S Gilpin and I. Davidson. Incorporating SAT solvers into hierarchical clustering algorithms : an efficient and flexible approach. In *KDD'11*, pages 1136–1144, 2011.

[14] M. Khiari, P. Boizumault, and B. Crémilleux. Constraint programming for mining n-ary patterns. In *CP'10*, volume 6308 of *LNCS*, pages 552–567, 2010.

[15] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[16] J.-P. Métivier, P. Boizumault, B. Crémilleux, M. Khiari, and S. Loudni. A constraint-based language for declarative pattern discovery. In *27th annual ACM Symposium on Applied Computing (SAC'12)*, pages 438–444, March 2012.

[17] L. De Raedt, T. Guns, and S. Nijssen. Constraint programming for itemset mining. In *KDD'08*, pages 204–212, 2008.

[18] C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *CP'05*, volume 3709 of *LNCS*, pages 827–831. Springer, 2005.

[19] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In P. Langley, editor, *ICML'00*, pages 1103–1110. M. Kaufmann, 2000.

[20] K. Wagstaff, C. Cardie, S. Rogers, and St. Schrödl. Constrained k-means clustering with background knowledge. In *ICML'01*, pages 577–584, 2001.

Agrégations de Among à l'aide de décompositions

Jean-Philippe Métivier **Samir Loudni**

Université de Caen – GREYC – (CNRS UMR 6072)
Campus II – Côte de Nacre – Boulevard du Maréchal Juin
14000 Caen – France
prenom.nom@unicaen.fr

Résumé

La décomposition de contraintes globales en un réseau Berge-acyclique permet, et cela sans perte de filtrage, d'intégrer simplement celles-ci dans des solveurs de contraintes (et cela de manière très efficace, cf. Regular). Dans ce papier, nous proposons d'utiliser la décomposition de la contrainte Among pour modéliser des agglomérations de contraintes Among, tout en préservant l'acyclicité du réseau issu de la décomposition.

1 Introduction

Les contraintes globales jouent un rôle majeur dans la modélisation et la résolution de problèmes de grande taille [6]. Celles-ci, en exploitant la structure du problème et en proposant des algorithmes de filtrage dédiés, permettent d'établir des cohérences fortes [1].Cependant l'intégration de contraintes globales dans un solveur de contraintes est une tâche complexe qui nécessite une grande expertise et une bonne connaissance du solveur.

La décomposition de contraintes globales en contraintes plus élémentaires facilite leurs mises en œuvre. Bien que facilement intégrables, toutes les décompositions ne permettent pas d'établir le même niveau de cohérence que le propagateur dédié.

Dans le cas où la décomposition forme un réseau Berge-acyclique, alors il est possible d'établir la cohérence de domaine via l'arc-cohérence sur le nouveau réseau de contraintes. De nombreux travaux proposent d'exploiter ce type de décomposition pour diverses contraintes globales : `Among` [1, 3], `Regular` [4], ...

Dans cet article, nous nous focalisons sur la contrainte `Among` [2]. Cette contrainte globale joue un rôle important dans la modélisation et la résolution de problèmes d'allocation de ressources. Dans ce type de problèmes, plusieurs contraintes `Among` peuvent être combinées afin de modéliser l'équilibrage de la charge de travail. Nous montrons, comment en utilisant la décomposition de la contrainte `Among`, il est alors possible d'établir la cohérence de domaine sur cette conjonction.

La suite de cet article est organisée de la façon suivante. La section 2 présente la contrainte `Among` ainsi que sa décomposition en un réseau Berge-acyclique. La section 3 donne un exemple d'équilibrage modélisé par une conjonction de contraintes `Among`, et montre comment décomposer celui-ci en un réseau Berge-acyclique. La section 4 décrit une méthode pour relaxer cette conjonction de contraintes. Finalement, la section 5 montre l'efficacité de cette approche au travers de plusieurs expériementations.

2 La contrainte Among

La contrainte globale `Among` est très utile pour modéliser de nombreux problèmes d'allocation de ressources, tels que le *staff rostering* ou le *car sequencing*.

2.1 La contrainte Among

La contrainte `Among` restreint le nombre de variables pouvant être assignées à des valeurs d'un ensemble donné :

Définition 1 (La contrainte `Among` [2])**.** *Soit* X=$\{X_1,X_2,...,X_n\}$ *un ensemble de variables et* S *un ensemble de valeurs. Notons* l *et* u *deux bornes telles que* $0 \leq l \leq u \leq n$*. La contrainte peut être définie par :*

$$\texttt{Among}(X, S, l, u) = \{(d_1, ..., d_n) \mid l \leq |\{d_i \in S, i \in [1..n]\}| \leq u\}$$

2.2 Décomposition Berge-acyclique de `Among`

Soit la contrainte `Among`($\{X_1,..,X_n\}$,S,l,u). Celle-ci peut être naturellement décomposée en un réseau de

†. Ce travail a été soutenu par l'Agence nationale de la Recherche, reference ANR-10-BLA-0214.

1. http ://www.emn.fr/z-info/sdemasse/gccat/

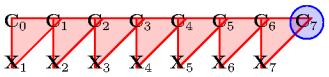


FIGURE 1 – Décomposition de la contrainte `Among` en un réseau Berge-acyclique.

contraintes ternaires et unaire Berge-acyclique [3] :

Définition 2 (Réseau de contraintes Berge-acyclique). *Un réseau de contraintes est Berge-acyclique ssi son graphe d'incidence est acyclique. Le graphe d'incidence* $G = (X \cup E, e_G)$ *d'un réseau de contraintes* $R = (X, E)$ *est un graphe tel que l'ensemble des sommets contient un sommet pour chaque variable et chaque contrainte et on retrouve une arête* $\{X_i, E_j\}$ *ssi* X_i *est dans la portée de* E_j.

Pour décomposer `Among`, il est nécessaire d'introduire un ensemble de $n+1$ variables supplémentaires C_i. Ces variables jouent le rôle de compteur et ont pour domaine $[0..i]$. Elles comptabilisent le nombre de fois qu'une valeur de S a été affectée à la séquence de variables $X_1,..,X_i$.

Une contrainte unaire (cercle en bleu dans la figure 1) est ajoutée pour contraindre la dernière variable compteur à prendre une valeur comprise entre l et u. Pour chaque variable X_i ($i \in \{1, \ldots, n\}$), une contrainte ternaire (C_{i-1},X_i,C_i) (triangle en rouge dans la figure 1) est également ajoutée. Cette contrainte impose que la valeur du compteur C_i est égale à $C_{i-1} + 1$ si la variable X_i est instanciée à une valeur de S, C_{i-1} sinon.

La figure 1 présente la décomposition d'une contrainte `Among` portant sur sept variables.

Comme indiqué dans [1], pour toute contrainte globale décomposable en un réseau Berge-acyclique, établir la cohérence d'arc sur le réseau décomposé est équivalent à établir la cohérence de domaine sur la contrainte globale elle-même.

3 Among et équilibrage

Dans de nombreux problèmes de *staff rostering*, l'équilibrage de la charge de travail joue un rôle important. En plus, des contraintes `Among` portant sur l'ensemble du planning (imposant des bornes sur le nombre de jours travaillés par un employé), on retrouve également un ensemble de contraintes `Among` supplémentaires portant sur des sous-intervalles (p.e. les semaines) répartissant la charge de travail.

Dans cette section, nous montrons comment il est possible de maintenir une cohérence de domaine sur une telle conjonction de contraintes, en exploitant de manière astucieuse les variables compteurs ajoutées par la décomposition.

3.1 Exemple introductif

Considérons l'exemple suivant extrait de l'instance GPOST [2], une instance réelle de Nurse Rostering Problems (NRP).

Exemple 1. *Sur un horizon de 28 jours, une infirmière à plein temps doit travailler au plus 18 jours (en équipe de jour D ou de nuit N). Afin d'assurer un équilibrage de la charge de travail, une infirmière doit travailler entre 4 et 5 fois par semaine.*

Il est possible de modéliser ce problème comme suit. Soit l'ensemble des jours J={1,2,...,28}. Pour chaque jour $j \in J$ est associé une variable X_j. Chaque variable prend ses valeurs sur l'ensemble des équipes possibles K={D,N,R}, avec R désignant un Repos.

Une première contrainte `Among` est utilisée pour contraindre le nombre de jours travaillés sur l'ensemble du planning. Puis pour chaque semaine, une contrainte `Among` est ajoutée pour assurer l'équilibrage. Ainsi, on obtient la modélisation suivante :

$\forall i \in [1..28],\ D_i = \{D, N, R\}$
`Among`($\{X_1, ..., X_{28}\}$,$\{D, N\}$,0,18)
`Among`($\{X_1, ..., X_7\}$,$\{D, N\}$,4,5)
`Among`($\{X_8, ..., X_{14}\}$,$\{D, N\}$,4,5)
`Among`($\{X_{15}, ..., X_{21}\}$,$\{D, N\}$,4,5)
`Among`($\{X_{22}, ..., X_{28}\}$,$\{D, N\}$,4,5)

Nous nommerons cette conjonction de contraintes `BalancedAmong`.

Définition 3 (La contrainte `BalancedAmong`). *Soit* X={$X_1, ..., X_n$} *un ensemble de variables, et* S *un ensemble de valeurs. Soit* X^i *(pour* $i \in [0..\frac{n}{k}]$*) l'ensemble de variables tel que* X^i={$X_{k\cdot(i-1)+1}, ..., X_{k\cdot i}$}. *Soit* l *et* u *(resp.* l^i *et* u^i*) les bornes associées à* X *(resp. à* X^i*). La contrainte est définie par :*

$$\texttt{BalancedAmong}(X, S, k, l, u, l^i, u^i) = \texttt{Among}(X, S, l, u) \ \wedge \ \bigwedge_{i=1}^{\frac{n}{k}} \texttt{Among}(X^i, S, l^i, u^i)$$

N.B.: Les valeurs des bornes l^i et u^i peuvent être différentes pour chaque X^i.

3.2 Décomposition « naïve »

Si on applique naïvement la décomposition à `BalancedAmong` sur l'exemple 1, on obtient le réseau de contraintes de la figure 2. Le premier constat qu'il est possible de dresser est le nombre relativement élevé de contraintes ternaires engendrées par les décompositions des différentes contraintes `Among`. Par aillleurs, le nouveau réseau de contraintes obtenu n'est plus Berge-acyclique. En effet, de nombreux cycles sont présents (par exemple $\{C_0^1, C_1^1, X_2, C_1^f, C_0^f, X_1, C_0^1\}$).

2. http ://www.cs.nott.ac.uk/~tec/NRP/GPost.html

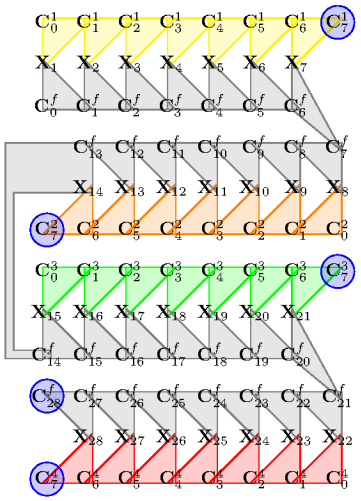


FIGURE 2 – Décomposition naïve du réseau de contraintes de l'exemple 1.

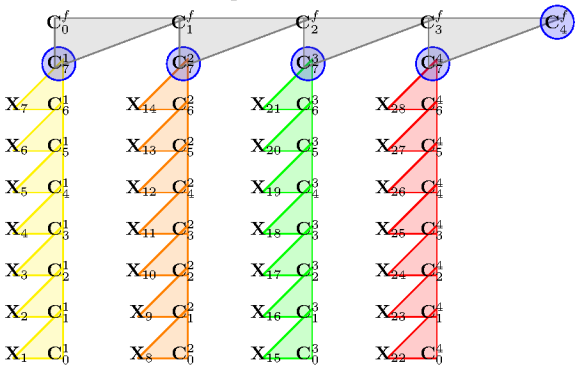


FIGURE 3 – Décomposition préservant l'acyclicité du réseau de contraintes de l'exemple 1.

3.3 Décomposition préservant l'acyclicité

Nous proposons d'exploiter les variables compteurs ajoutées par la décomposition pour d'une part rendre le réseau acyclique et d'autre part pour réduire le nombre de contraintes ternaires.

Pour chaque contrainte $\texttt{Among}(X^i, S, l^i, u^i)$, la dernière variable compteur C^i_k correspond au nombre de fois qu'une valeur de S est affectée à une variable de X^i. Ainsi, pour connaître le nombre de variables de X prenant leur valeur dans S, il suffit de sommer les C^i_k de tous les X^i (car $\sqcup_{i\in[0,n/k]} X^i = X$). La contrainte globale somme est elle-même décomposable en un ensemble de contraintes ternaires.

La figure 3 illustre cette nouvelle décomposition utilisant une contrainte globale somme.

4 Relaxation

Pour relaxer une conjonction des contraintes `Among`, il suffit de pondérer chacune des contraintes unaires de la décomposition. Il est ainsi possible de combiner plusieurs sémantiques de violation.

La table 1 montre un exemple de fonctions de coût pénalisant tout écart δ sur un sous-ensemble (resp. l'intégralité) par un coût quadratique δ^2 (resp. linéaire $\delta \times 100$).

C^1_7	f
0	16
1	9
2	4
3	1
4	0
5	0
6	1
7	4

C^2_7	f
0	16
1	9
2	4
3	1
4	0
5	0
6	1
7	4

C^3_7	f
0	16
1	9
2	4
3	1
4	0
5	0
6	1
7	4

C^4_7	f
0	16
1	9
2	4
3	1
4	0
5	0
6	1
7	4

C^f_4	...	16	17	18	19	20	21	...
f	...	0	0	0	100	200	300	...

TABLE 1 – Sémantiques de violation associées à la décomposition du réseau de contraintes de l'exemple 1.

5 Expérimentations

Nous avons implémenté dans `ToulBar2 0.9.5`[3] les deux versions décomposées de la conjonction de contraintes `Among` ainsi que leurs versions relaxées. Les expériences ont été conduites sur un Core2Duo E8400 (2.83Ghz) avec 4GB de RAM. Les résultats reportés dans cette section (temps de calcul et nombre de backtracks) sont des sommes sur 100 instances.

5.1 Problème « pur »

Dans ce premier jeu d'expérimentations, pour chaque valeur du paramètre w, nous avons généré aléatoirement 100 instances. Chaque instance contient un ensemble de fonctions de coût unaire et une contrainte globale (`BalancedAmong`) agglomérant une conjonction de contraintes `Among`. Chaque instance contient $7 \times w$ variables ayant pour domaine l'ensemble {0,1,2}. Les contraintes unaires sont réparties sur 90% des variables, et contraignent celles-ci à prendre une valeur donnée contre un coût compris entre 1 et 10.

La contrainte `BalancedAmong` impose que les valeurs 0 et 1 soient affectées au plus $\lfloor 4.5 \times w \rfloor$ fois sur l'ensemble du planning. Pour chaque ensemble de 7 variables ($[X_1,...,X_7]$, $[X_8,...,X_{14}]$, ...) la limite est fixée entre 4 et 5 fois. Tout écart δ sur l'ensemble des variables est pénalisé par un coût linéaire de $\delta \times 100$ et chaque écart δ sur un ensemble de 7 variables est pénalisé par un coût quadratique δ^2.

Nous avons comparé trois implémentations :

1. la décomposition en un réseau Berge-acyclique de `BalancedAmong` combiné avec un ensemble de fonctions unaires (notée *Balanced*) ;
2. la décomposition de `BalancedAmong` à l'aide de contraintes `Among` décomposées et un ensemble de fonctions unaires (notée *AmongDec*) ;

3. https ://mulcyber.toulouse.inra.fr/projects/toulbar2

	Balanced		*AmongDec*		*AmongMono*	
w	#bt	time (s)	#bt	time (s)	#bt	time (s)
2	0	0.8	194	1.1	643	1.7
4	0	0.9	940	2.7	13'445	5.1
6	0	1.9	2'465	5.4	63'980	12.8
8	0	2.8	4'646	10.3	584'387	36.7
10	0	4.1	8'762	19.5	838'623	90.6

TABLE 2 – Comparaison des trois implémentations sur un problème « pur ».

	Balanced		*AmongDec*		*AmongMono*	
w	success	time (s)	success	time (s)	success	time (s)
2	100%	0.9	100%	1.3	100%	16.3
4	100%	1.9	100%	2.7	100%	77.9
6	100%	3.7	100%	5.8	100%	492.8
8	100%	6.1	100%	11.0	92%	17'937.7
10	100%	11.3	100%	29.8	28%	81'421.7

TABLE 3 – Comparaison des trois implémentations avec des contraintes `Regular` additionnelles.

3. une contrainte `BalancedAmong` exprimée à l'aide de `Among` monolithiques dans `Choco 2.1.3`[4] et des fonctions de coût unaire[5] (notée *AmongMono*).

La recherche est effectué sans limite de temps de calcul et sans utiliser d'option particulière.

Le tableau 2 montre clairement que la première implémentation permet d'obtenir des gains très substentiels à la fois en nombre de backtracks qu'en temps de résolution.

5.2 Avec des contraintes Regular additionnelles

Nous avons également généré un second jeu d'instances contenant 50% de fonctions de coût unaire (fixées de la même façon que précédemment) et une contrainte `BalancedAmong`. Pour contraindre d'avantage les instances, nous avons ajouté deux contraintes `Regular` : la première contrainte permet d'imposer un ordre sur l'enchaînement des valeurs et la seconde contrainte permet de limiter le nombre de fois successives que la valeur 1 peut être prise.

Le tableau 3 compare à nouveau les résultats des trois implémentations. La première colonne indique le pourcentage d'instances résolues en 1000 secondes, et la seconde le temps de calcul total (1000 secondes sont reportées lorsque la résolution d'une instance s'est achevés par épuisement du temps de calcul).

Une fois de plus, l'utilisation de la décomposition Berge-acyclique offre des gains substentiels.

5.3 Retour sur l'exemple : GPOST

Dans ce denier jeu d'expériences, nous avons mesuré l'apport de `BalancedAmong` sur les deux instances `GPOST`s. Nous avons utilisé la même modilisation que dans [5] et la même méthode de résolution : `VNS/LDS+CP` (et `RandomNurse` comme heuristique de choix de voisinages). Pour chaque instance, nous avons effectué 100 essais (avec un temps d'exécution maximal de 1800 secondes).

		Balanced			*AmongDec*		
Instance	Opt	succ.	avg	time (s)	succ.	avg	time (s)
`GPOSTA`	5	18	15.2	677.9	12	21.4	858.3
`GPOSTB`	3	9	44.9	732.8	5	62.1	463.8

TABLE 4 – Comparaison sur les instances `GPOST`s.

Le tableau 4 présent le nombre de succés – le nombre de fois que l'optimum est atteint –, la moyenne de la qualité des solutions retournées, et le temps moyen pour trouver une solution de coût optimal.

L'utilisation de `BalancedAmong` permet de grandement augmenter le nombre de succés (dans la cas de `GPOST-B`, celui-ci est quasiment doublé).

6 Conclusions et futurs travaux

Dans ce papier, nous avons présenté, comment en utilisant la décomposition de `Among`, il est possible d'établir la cohérence de domaine sur une conjonction particulière de `Among`. Nous avons aussi montré, expérimentalement, l'apport de cette nouvelle contrainte.

Comme futurs travaux, il serait utile de tester l'apport de `BalancedAmong` sur des problèmes de *rostering* ou de *car sequencing*. Il serait aussi intéressant d'étudier l'apport de la décomposition sur d'autres conjonctions de `Among` ou d'autres contraintes globales comme `Sum` ou `Regular`.

Références

[1] N. Beldiceanu, M. Carlsson, R. Debruyne, and T. Petit. Reformulation of global constraints based on constraints checkers. *Constraints*, 10(4) :339–362, 2005.

[2] N. Beldiceanu and E. Contejean. Introducing global constraints in chip. *Journal of Mathematical and Computer Modelling*, 20(12) :97–123, 1994.

[3] C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Among, common and disjoint constraints. In *CSCLP*, pages 29–43, 2005.

[4] C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Slide : A useful special case of the card-path constraint. In *ECAI*, pages 475–479, 2008.

[5] J-P. Métivier, P. Boizumault, and S. Loudni. Solving nurse rostering problems using soft global constraints. In *CP*, pages 73–87, 2009.

[6] H. Simonis. Models for global constraint applications. *Constraints*, 12(1) :63–92, 2007.

4. http ://www.emn.fr/z-info/choco-solver/
5. Modelisée à l'aide de contraintes réifiées.

Un modèle booléen pour l'énumération des siphons et des pièges minimaux dans les réseaux de Petri

Faten Nabli **François Fages** **Thierry Martinez** **Sylvain Soliman**

EPI CONTRAINTES
INRIA Paris-Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105,
78153 LE CHESNAY CEDEX - FRANCE
{Faten.Nabli, Francois.Fages, Thierry.Martinez, Sylvain.Soliman}@inria.fr

Résumé

Les réseaux de Petri sont un formalisme simple pour modéliser les calculs concurrents. Récemment, ils se sont révélés être un outil puissant de modélisation et d'analyse des réseaux de réactions biochimiques, en comblant le fossé entre les modèles purement qualitatifs et quantitatifs. Les réseaux biologiques peuvent être larges et complexes, ce qui rend leur analyse difficile. Dans cet article, nous nous concentrons sur deux propriétés structurelles des réseaux de Petri : les siphons et les pièges, qui nous apportent des informations sur la persistance de certaines espèces biochimiques. Nous présentons deux méthodes pour énumérer les siphons et les pièges minimaux d'un réseau de Petri en itérant la résolution d'un problème booléen, interprété comme un programme SAT ou PLC(B). Nous comparons les performances de ces méthodes avec un algorithme dédié qui représente l'état de l'art dans la communauté des réseaux de Petri. Nous montrons que les programmes SAT et PLC(B) sont plus efficaces. Nous analysons pourquoi ces programmes sont si performants sur les modèles de l'entrepôt de modèles biomodels.net et nous proposons des instances difficiles pour le problème de l'énumération des siphons minimaux.

Abstract

Petri-nets are a simple formalism for modeling concurrent computation. Recently, they have emerged as a powerful tool for the modeling and analysis of biochemical reaction networks, bridging the gap between purely qualitative and quantitative models. These networks can be large and complex, which makes their study difficult and computationally challenging. In this paper, we focus on two structural properties of Petri-nets, siphons and traps, that bring us information about the persistence of some molecular species. We present two methods for enumerating all minimal siphons and traps of a Petri-net by iterating the resolution of a boolean model interpreted as either a SAT or a CLP(B) program. We compare the performance of these methods with a state-of-the-art dedicated algorithm of the Petri-net community. We show that the SAT and CLP(B) programs are both faster. We analyze why these programs perform so well on the models of the repository of biological models biomodels.net, and propose some hard instances for the problem of minimal siphons enumeration.

1 Introduction

Les réseaux de Petri ont été introduits dans les années 60 comme un formalisme simple pour décrire et analyser les systèmes concurrents, asynchrones, non déterministes et éventuellement distribués.

L'utilisation des réseaux de Petri pour représenter les modèles de réactions biochimiques, en formalisant les espèces moléculaires par les places et les réactions par les transitions, a été introduite assez tardivement dans [19] , en proposant certains concepts et outils des réseaux de Petri pour l'analyse de ces réseaux biochimiques. Dans [20], un programme logique avec contraintes sur domaines finis (PLC(DF)) est proposé pour le calcul des P-invariants. Ces invariants fournissent des lois de conservation structurelles qui peuvent être utilisées pour réduire la dimension des équations différentielles ordinaires (EDO) associées à un modèle de réactions avec expressions cinétiques.

Dans ce papier, nous considérons les concepts de siphons et pièges des réseaux de Petri. Ces structures sont liées au comportement dynamique du réseau : la vérification de l'accessibilité (est-ce que le système peut atteindre un état donné) et à la vivacité (ab-

sence de blocages). Ces propriétés ont été déjà utilisées pour l'analyse des réseaux métaboliques [24]. Un programme linéaire en nombres entiers est proposé dans [3] et un algorithme dédié est décrit dans [5] pour les calculer.

Un siphon est un ensemble de places qui, une fois non marqué, le reste. Un piège est un ensemble de places qui, une fois marqué, ne peut jamais perdre tous ses jetons. Un exemple typique de siphon est un ensemble de métabolites qui sont progressivement réduits au cours d'une famine ; un exemple typique de piège est l'accumulation de métabolites qui sont produits au cours de la croissance d'un organisme.

Dans cet article, après quelques préliminaires sur les réseaux de Petri, les siphons et les pièges, nous donnons un modèle booléen simple de ces propriétés. Nous décrivons deux méthodes pour énumérer l'ensemble des siphons et des pièges minimaux et nous les comparons avec un algorithme dédié qui représente l'état de l'art [5], sur un banc d'essai composé à la fois du dépôt Petriweb [8] et de l'entrepôt de modèles biomodels.net [13]. Dans la dernière section, nous expliquons pourquoi les programmes proposés sont très performants sur les modèles biochimiques de l'entrepôt Biomodels.net et proposons certaines instances difficiles pour le problème de l'énumération des siphons minimaux.

2 Préliminaires

2.1 Réseaux de Petri

Un réseau de Petri PN est un graphe biparti orienté $PN = (P, T, W)$, où P est un ensemble fini de sommets appelés places, T est un ensemble fini de sommets (disjoint de P) appelés transitions et $W : ((P \times T) \cup (T \times P)) \to \mathbf{N}$ représente un ensemble d'arcs orientés et pondérés par des entiers (le poids zéro représente l'absence d'arc). Un marquage d'un graphe de réseau de Petri est une fonction $m : P \to \mathbf{N}$, qui affecte un certain nombre de jetons à chaque place. Un réseau de Petri marqué est un 4-tuple (P, T, W, m_0) où (P, T, W) est un réseau de Petri et m_0 est un marquage initial.

L'ensemble des prédécesseurs (resp. successeurs) d'une transition $t \in T$ est l'ensemble des places ${}^\bullet t = \{p \in P \mid W(p,t) > 0\}$ (resp. $t^\bullet = \{p \in P \mid W(t,p) > 0\}$). De même, l'ensemble des prédécesseurs (resp. successeurs) d'une place $p \in P$ est l'ensemble de transitions ${}^\bullet p = \{t \in T \mid W(t,p) > 0\}$ (resp. $p^\bullet = \{t \in T \mid W(p,t) > 0\}$).

Pour tous marquages $m, m' : P \to \mathbf{N}$ et toute transition $t \in T$, il y a une étape de transition $m \xrightarrow{t} m'$, si et seulement si pour tout $p \in P$, $m(p) \geq W(p,t)$ et $m'(p) = m(p) - W(p,t) + W(t,p)$.

Cette notation reste valable pour une séquence de transitions $\sigma = (t_0 \dots t_n)$ en écrivant $m \xrightarrow{\sigma} m'$ si $m \xrightarrow{t_0} m_1 \xrightarrow{t_1} \dots \xrightarrow{t_{n-1}} m_n \xrightarrow{t_n} m'$ pour les marquages $m_1, \dots, m_n$.

La représentation classique d'un modèle de réactions par un réseau de Petri consiste à associer aux espèces chimiques des *places* et aux réactions des *transitions*.

Exemple 1. *Le modèle réactionnel de la réaction enzymatique de Michaelis-Menten A + E <=> AE => B + E correspond au réseau de Petri de la Figure 1.*

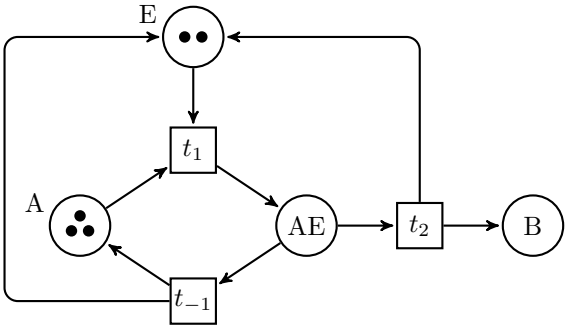


FIGURE 1 – Modèle biochimique de l'Exemple 1, représenté comme un réseau de Petri avec un marquage validant t_1.

2.2 Siphons et Pièges

Soit $PN = (P, T, W)$ un réseau de Petri.

Définition 1. *Un piège est un ensemble non vide de places $P' \subseteq P$ tels que ses successeurs sont aussi des prédécesseurs : $P'^\bullet \subseteq {}^\bullet P'$.*

Un siphon est un ensemble non vide de places $P' \subseteq P$ tel que ses prédécesseurs sont aussi des successeurs. ${}^\bullet P' \subseteq P'^\bullet$.

Remarquons qu'un siphon dans PN est un piège dans le réseau de Petri dual obtenu en inversant la direction de tous les arcs de PN. Notons aussi que puisque les successeurs (resp. prédécesseurs) d'une union sont l'union des successeurs (resp. prédécesseurs), l'union de deux siphons (resp. pièges) est un siphon (resp. piège).

Les propositions suivantes montrent que les pièges et les siphons donnent une caractérisation structurelle de certaines propriétés dynamiques des marquages.

Proposition 1. *[18] Pour tout sous-ensemble de places $P' \subseteq P$, P' est un piège si et seulement si pour tout marquage $m \in \mathbf{N}^P$ tel que $m_p \geq 1$ pour certaines*

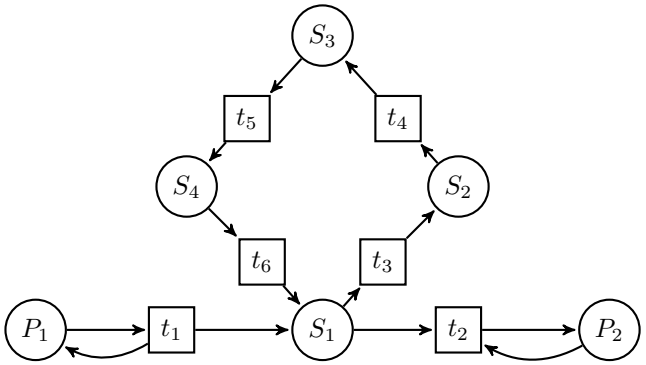


FIGURE 2 – Réseau de Petri de [24] modélisant la croissance de la plante de la pomme de terre.

places $p \in P'$, et pour tout marquage $m' \in \mathbf{N}^P$ tel que $m \xrightarrow{\sigma} m'$ pour une séquence de transitions σ, il existe une place $p' \in P'$ telle que $m'_{p'} \geq 1$.

Proposition 2. *[18] Pour tout sous-ensemble de places $P' \subseteq P$, P' est un siphon si et seulement si pour tout marquage $m \in \mathbf{N}^P$ avec $m_p = 0$ pour tout $p \in P'$, et tout marquage $m' \in \mathbf{N}^P$ tel que $m \xrightarrow{\sigma} m'$ pour une séquence de transitions σ, nous avons $m'_{p'} = 0$ pour tout $p' \in P'$.*

Un exemple d'utilisation des siphons et des pièges a été introduit dans [24] : la plante de la pomme de terre produit l'amidon et l'accumule pendant la croissance pour ensuite le consommer après la récolte. Le réseau de Petri correspondant est représenté par la Figure 2, où S_1 représente le glucose-1-phosphate, S_2 est UDP-glucose et S_3 est l'amidon [21]. Dans ce modèle, l'une des deux branches, soit la branche produisant l'amidon (t_3 et t_4), soit la branche le consommant (t_5 et t_6), est opérationnelle. P_1 et P_2 représentent des métabolites externes.

Il est facile d'observer que l'ensemble $\{S_2, S_3\}$ est un piège lorsque t_3 et t_4 sont opérationnelles et que $\{S_3, S_4\}$ est un siphon lorsque t_5 et t_6 sont opérationnelles : une fois un jeton arrive dans S_3, aucune transition ne peut être franchie et le jeton y reste indépendamment de l'évolution du système. D'autre part, une fois le dernier jeton est consommé de S_3 et S_4, aucune transition ne pourra générer un nouveau jeton dans ces places qui demeureront vides.

Dans beaucoup de cellules contenant de l'amidon, ce dernier et certains de ces prédécesseurs forment des pièges, alors que l'amidon et certains successeurs forment des siphons. En effet, soit la branche produisant l'amidon, soit la branche qui le consomme, est opérationnelle et ceci est réalisé par une inactivation complète des enzymes appropriées.

2.3 Minimalité

Un siphon (resp. un piège) est minimal s'il ne contient pas d'autre siphon (resp. piège). Malgré la stabilité des siphons et des pièges par union, notons que les siphons minimaux ne forment pas un ensemble générique de tous les siphons.

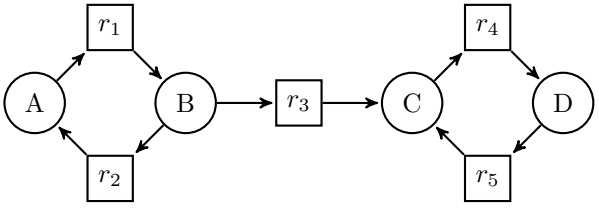


FIGURE 3 – Réseau de Petri de l'exemple 2.

Exemple 2. *Dans le réseau de Petri de la Figure 3, $\{A, B\}$ est un siphon minimal : ${}^\bullet\{A, B\} = \{r_1, r_2\} \subset \{A, B\}^\bullet = \{r_1, r_2, r_3\}$. $\{C, D\}$ est un piège minimal : $\{C, D\}^\bullet = \{r_4, r_5\} \subset {}^\bullet\{C, D\} = \{r_3, r_4, r_5\}$.*

Un siphon est générique s'il ne peut pas être représenté sous la forme d'une union d'autres siphons [16]. Un siphon minimal est un siphon générique, mais un siphon générique n'est pas forcément minimal. Prenons le cas de l'exemple 2, les siphons sont $\{A, B\}$ et $\{A, B, C, D\}$: mais seul le premier est minimal, le second ne pourra pas être écrit comme une union de siphons minimaux.

Une motivation pour étudier les siphons minimaux est qu'ils procurent une condition suffisante pour la non-existence de blocages. En effet, on a prouvé que dans un réseau de Petri bloqué (i.e. où aucune transition ne peut être franchie), toutes les places non-marquées forment un siphon [2]. Ainsi, l'approche basée sur les siphons pour la détection des blocages vérifie si le réseau contient un siphon propre (un siphon est propre si l'ensemble de ses prédécesseurs est strictement inclus dans l'ensemble de ses successeurs) peut devenir non marqué par une séquence de franchissement. Un siphon propre ne devient pas non marqué s'il contient un piège initialement marqué. Si un tel siphon est identifié, le marquage initial est modifié par la séquence de franchissement et la vérification continue pour les siphons restants jusqu'à ce qu'un blocage soit identifié ou jusqu'à ce que la vérification est terminée. Considérer uniquement l'ensemble des siphons minimaux est suffisant puisque si un siphon devient non marqué durant l'analyse alors au moins un siphon minimal doit aussi être non marqué.

D'autres liens avec les propriétés comportementales de vivacité sont établis dans [9].

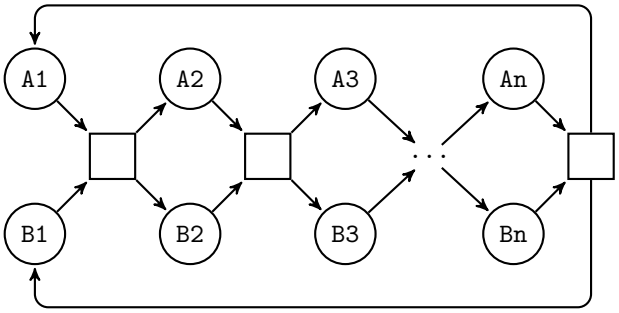


FIGURE 4 – Réseau de Petri du modèle de l'exemple 3.

2.4 Complexité

Décider si un réseau de Petri contient un siphon ou un piège et en donner un s'il existe est polynomial [4]. Cependant, le problème de décision de l'existence d'un siphon minimal contenant une place donnée est NP-difficile [22]. De plus, il peut y avoir un nombre exponentiel de siphons et de pièges dans un réseau de Petri comme le montre l'exemple suivant.

Exemple 3. *Dans le réseau de Petri définie par les équations A1 + B1 => A2 + B2, A2 + B2 => A3 + B3, ..., An + Bn => A1 + B1. et représenté par la Figure 4, il y a 2^n siphons minimaux et 2^n pièges minimaux, chacun contenant soit Ai soit Bi mais pas les deux en même temps, pour tout i.*

3 Modèle booléen

Dans la littérature, plusieurs algorithmes ont été proposé pour le calcul des siphons et des pièges minimaux d'un réseau de Petri. Puisque un siphon dans un réseau de Petri N est un piège dans le réseau dual N', il suffit de traiter les siphons, les pièges sont obtenus par dualité. Certains algorithmes sont basés sur les inégalités [16], les équations logiques [10, 14], ou des approches algébriques [12]. Des méthodes plus récentes ont été présentées dans [4, 5].

Le problème de recherche d'un siphon peut être naturellement décrit par un modèle booléen représentant l'appartenance ou non de chaque place au siphon recherché. L'énumération de tous les siphons peut être codée comme une procédure de recherche itérative, similaire à l'approche branch-and-bound grâce à une propriété donnée ci-dessous.

Pour un réseau de Petri contenant n places et m transitions, un siphon S est un ensemble de places tel que ses prédécesseurs sont aussi successeurs. S peut être représenté par un vecteur $\vec{V}$ de $\{0,1\}^n$ tel que pour tout $i \in \{1,2,..,n\}$, $V_i = 1$ si et seulement si $p_i \in S$.

La contrainte de siphon peut être formulée comme suit :

$$\forall i, V_i = 1 \Rightarrow (\forall t \in T, t \in {}^\bullet p_i \Rightarrow t \in (\cup_{V_j=1}\{p_j\})^\bullet).$$

Cette contrainte est équivalente à :

$$\forall i, V_i = 1 \Rightarrow {}^\bullet p_i \subseteq (\bigcup_{V_j=1}\{p_j\})^\bullet$$

ce qui peut être écrit sous la forme :

$$\forall i, V_i = 1 \Rightarrow \bigwedge_{t \in {}^\bullet p_i}(\bigvee_{p_j \in t^\bullet} V_j = 1).$$

Finalement, afin d'exclure l'ensemble vide, la contrainte suivante est ajoutée

$$\bigvee_i V_i = 1.$$

Énumérer uniquement les siphons minimaux (vis-à-vis de l'inclusion ensembliste) peut être assuré par la stratégie de recherche et l'ajout de nouvelles contraintes.

La stratégie trouve un siphon minimal vis-à-vis de l'ordre d'inclusion ensembliste et une nouvelle contrainte est ajoutée à chaque fois qu'un tel siphon est trouvé afin d'interdire les siphons le contenant dans la suite de la recherche.

Dans une approche précédente [17], pour la méthode basée sur la programmation avec contraintes, l'ordre d'inclusion était assuré par un étiquetage (labeling) sur une variable cardinalité dans un ordre croissant. Étiqueter directement sur les variables booléennes, par valeur croissante (d'abord 0, puis 1), se révèle être plus efficace, plus facile à appliquer, et garantit que les siphons sont trouvés dans l'ordre d'inclusion ensembliste, grâce à la proposition suivante.

Proposition 3. *Soit un arbre binaire tel que, dans chaque nœud instanciant une variable X l'arc gauche poste la contrainte $X = 0$ et l'arc droit poste la contrainte $X = 1$, alors pour toutes les feuilles distinctes A et B, la feuille A est à gauche de la feuille B seulement si l'ensemble représenté par B n'est pas inclus dans l'ensemble représenté par A (c'est-à-dire, il existe une variable X telle que $X_B > X_A$, où X_A et X_B dénotent les valeurs instanciées à X dans le trajet menant à A et B respectivement).*

Preuve. A et B ont au moins un nœud ancêtre en commun instanciant une variable X. Si la feuille A est à gauche de la feuille B, l'arc menant à A est à gauche avec la contrainte $X = 0$ et l'arc menant à B est à droite avec la contrainte $X = 1$, par conséquent $X_B > X_A$. □

À chaque fois qu'un siphon (S_i) est trouvé, la contrainte $\bigvee_{i|S_i=1} V_i = 0$ doit être ajoutée au modèle pour garantir que les sur-ensembles de ce siphon ne seront pas énumérés.

4 Algorithmes

Cette section décrit deux implémentations du modèle précédent ainsi que deux stratégies de recherche, une utilisant une procédure SAT itérée et l'autre basée sur la programmation avec contraintes.

4.1 Algorithme SAT itéré

Le modèle booléen peut être directement interprété en utilisant un solveur SAT pour vérifier l'existence d'un siphon ou un piège.

Nous utilisons sat4j, la bibliothèque de satisfiabilité et d'optimisation booléenne pour Java qui fournit une collection de solveurs SAT efficaces. Elle inclue une implémentation des spécifications MiniSAT en Java.

Pour le réseau de Petri de la figure 1 représentant la réaction enzymatique de l'exemple 1, nous avons le codage suivant : chaque ligne est une liste de variables séparées par des espaces, elle représente une clause. Une valeur positive signifie que la variable correspondante est sous la forme positive (donc 2 signifie V_2), et une valeur négative signifie la négation de la variable (donc -3 signifie $-V_3$). Dans cet exemple, les variables 1, 2, 3 and 4 correspondent respectivement à E, A, AE et B. Dans la première itération, le problème est de résoudre les clauses suivantes :
-2 3
-3 1 2
-1 3
-1 3
-4 3
Le problème est satisfiable avec les valeurs : -1, 2, 3, -4 ce qui signifie que $\{A, AE\}$ est un siphon minimal. Pour assurer la minimalité, la clause -2 -3 est ajoutée et le programme itère une autre fois. Le problème est satisfiable avec les valeurs 1, -2, 3, -4, ce qui signifie que $\{E, AE\}$ est aussi un siphon minimal. Une nouvelle clause est ajoutée précisant que soit E soit AE n'appartient pas au siphon et aucune affectation des variable ne peut satisfaire le problème.

Ainsi, ce modèle contient 2 siphons minimaux : $\{A, AE\}$ et $\{E, AE\}$.

L'enzyme E est une protéine qui catalyse (i.e., augmente la vitesse de) la transformation du substrat E en produit B. L'enzyme est conservée dans une réaction chimique.

Les résultats obtenus avec cette procédure SAT itérée sont très bons, comme le détaille la section 5.

4.2 Algorithme PLC(B)

La recherche de siphons peut aussi être implémentée avec un Programme Logique avec Contraintes Booléennes (PLC(B)). Nous utilisons GNU-Prolog [7] pour l'efficacité de ses propagateurs de contraintes booléennes.

La stratégie d'énumération est une variation du *branch-and-bound*, où, à chaque fois qu'un nouveau siphon est trouvé, la recherche doit trouver un non sur-ensemble de ce siphon.

Nous avons essayé deux variantes de branch-and-bound : avec et sans relance. Dans le branch-and-bound avec relance, il se révèle essentiel de choisir une méthode de sélection des variables diversifiante. En effet, les méthodes d'énumération avec un ordre fixe de variables accumulent les échecs en essayant toujours d'énumérer les mêmes ensembles en premier, qui sont élagués plus tard par les contraintes de non sur-ensembles. L'arbre devient ainsi de plus en plus dense à chaque itération puisque plusieurs échecs précédents sont explorés à nouveau. Une sélection aléatoire des variables assure une bonne diversité, tout comme une méthode qui énumère d'abord sur les variables correspondant aux places figurant dans les siphons déjà trouvés.

Branch-and-bound sans relance donne de meilleurs performances à condition de prendre le soin de poster chaque contrainte de non sur-ensembles une seule fois (les reposer toutes à chaque backtrack est inefficace). Cette stratégie est implémentée de la manière suivante : à chaque fois qu'un siphon est trouvé, le chemin menant à cette solution est mémorisé, puis la recherche est entièrement défaite dans le but d'ajouter au modèle une nouvelle contrainte de non sur-ensemble, puis le chemin mémorisé est rejoué pour poursuivre la recherche au point où elle a été arrêtée. La figure 5 montre l'arbre de recherche qui est développé, il est beaucoup plus satisfaisant que celui obtenu avec relance.

Dans une phase de post-traitement, l'ensemble des siphons minimaux peut être filtré afin de ne garder que les siphons minimaux qui contiennent un ensemble donné de places, pour résoudre le problème NP-difficile sus-mentionné. Notons que poster l'inclusion de l'ensemble sélectionné de places en premier n'assurerait pas que les siphons trouvés sont minimaux vis-à-vis de l'inclusion ensembliste.

5 Évaluation

5.1 Banc d'essai Petriweb

Notre premier banc d'essai de réseaux de Petri est le dépôt Petriweb [8].

Les instances les plus difficiles sont des études de cas dans des processus de raffinement :

– *"Transit" case study - transit1 process*, identifiant 1454 ;

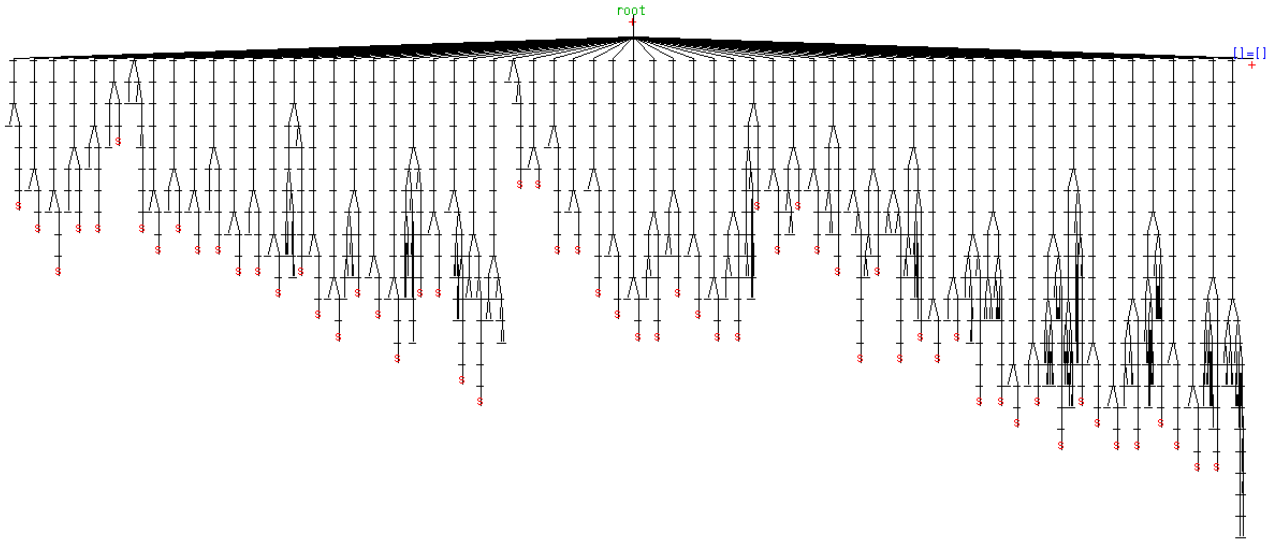


FIGURE 5 – Arbre de recherche développé avec la stratégie de backtrack sans relance pour le calcul de tous les siphons minimaux du réseau 1454 de PetriWeb.

- *"Transit" case study - transit2 process*, identifiant 1479 ;
- *"Transit" case study - transit4 process*, identifiant 1516 ;

5.2 Banc d'essai Biomodels.net

Nous considérons également l'entrepôt Biomodels.net de modèles de réactions biochimiques [13] ainsi que d'autres modèles complexes. Les modèles les plus grands sont comme les suivants :

- la carte de Kohn [11, 1] représente une carte de 509 espèces et 775 interactions moléculaires qui régissent le cycle cellulaire des mammifères et le mécanisme de réparation de l'ADN ;
- le modèle 175 qui représente les réponses au Ligand du réseau de signalisation de l'ErbB ;
- le modèle 205 qui représente la simulation de la régulation de l'endocytose de l'EGFR et la signalisation EGFR-ERK ;
- le modèle 239 qui représente un modèle cinétique du réseau de la sécrétion d'insuline stimulée par le glucose des cellules bêta du pancréas.

5.3 Résultats et Comparaisons

Dans [17], l'approche PLC a été comparée à un modèle linéaire en nombres entiers (PLNE) [3] sur le dépôt de Biomodels.net elle s'est révélée au moins trois fois plus rapide que l'énumération PLNE en utilisant un solveur CPLEX. Cet approche PLC implémente le modèle booléen décrit dans la section 4. Dans cette section, nous comparons deux implémentations de ce modèle booléen à l'algorithme dédié de l'état de l'art que les mêmes auteurs avaient introduit dans [5].

L'algorithme dédié utilise récursivement une procédure de partitionnement pour réduire le problème original à plusieurs sous-problèmes plus simples. Chaque sous-problème possède des contraintes spécifiques supplémentaires sur les places par rapport au problème d'origine. Dans [5] la correction, la convergence et la complexité de l'algorithme sont montrées. L'évaluation expérimentale de la performance est également traitée.

Ces algorithmes peuvent être appliqués pour énumérer les siphons minimaux, les siphons minimaux par rapport à une place ou aussi les siphons minimaux par rapport à un sous-ensemble donné de places.

Les tables 1 et 2 fournissent les performances sur les modèles de Petriweb et Biomodels.net. Dans la table 1, nous donnons les temps de calcul de tous les siphons minimaux des modèles présentés dans les sections 5.1 et 5.2. Pour chacun de ces exemples, nous fournissons le nombre de siphons minimaux, le nombre de places, le nombre de transitions et les temps de calcul en utilisant l'algorithme dédié, l'algorithme SAT et le programme PLC.

Toutes les durées sont en ms. Les temps de calcul sont obtenu en utilisant un PC avec un processeur intel Core 2.20 GHz et 8 Go de mémoire.

Dans la table 2, nous considérons l'ensemble des modèles mis à disposition dans les dépôts de Biomodels.net et de Petriweb. Pour que ces temps totaux aient un sens, nous avons retiré le modèle numéro 175 de biomodels.net et le réseau numéro 1516 de Petriweb

car les temps mesurés sur ces deux modèles sont à la fois très grands et non représentatifs du cas général : on se reportera à la table 1 pour les temps obtenus sur ces deux modèles. Pour chaque base de données, nous donnons le nombre total de modèles, le nombre minimal et le nombre maximal de siphons trouvés et la moyenne des nombres des siphons trouvés dans tous les modèles. Nous donnons également la taille minimale et la taille maximale des siphons trouvés ainsi que la moyenne des tailles sur tous les modèles. Finalement, nous fournissons le temps de calcul total qui n'est autre que la somme des temps de calcul de chaque modèle.

Sur toutes les instances, notamment les plus larges, le codage SAT est très efficace. Il convient parfaitement à la taille du réseau ainsi qu'au nombre de siphons minimaux. L'algorithme dédié est souvent moins efficace avec au moins un ordre de grandeur de différence, sauf pour une seule instance, le modèle numéro 1516 de biomodels.net, pour laquelle l'algorithme dédié est environ deux fois plus rapide : cette instance est singulière par son grand nombre de siphons minimaux. Le codage CLP a également une meilleur performance que l'algorithme dédié, mais il semble manipuler moins facilement des réseaux de grande taille tels que la carte de Kohn, et est très lent sur le modèle 1516.

6 Instances difficiles

MiniSAT dépasse en rapidité l'algorithme spécialisé d'au moins un ordre de grandeur et le temps de calcul est étonnamment court sur nos exemples pratiques. Même si le modèle est très grand, par exemple la carte de Kohn du contrôle du cycle cellulaire avec 509 espèces et 775 réactions, le temps de calcul demeure infime. Pourtant, cette énumération de tous les siphons minimaux résout le problème de décision de l'existence d'un siphon minimal contenant un ensemble donné de places qui a été prouvé NP-difficile dans [23] et la question est : pourquoi le calcul des siphons minimaux est-il si facile dans les grands réseaux biochimiques ou dans les réseaux de PetriWeb ?

Une façon d'aborder cette question est de considérer la preuve de NP-difficulté par réduction du problème 3-SAT et le phénomène de transition de phase dans 3-SAT. La probabilité qu'un problème 3-SAT aléatoire soit satisfiable suit une transition de phase aigue quand la densité α du nombre de clauses sur le nombre de variables est au voisinage de 4.26 [15, 6], allant de la satisfiabilité vers l'insatisfiabilité avec une probabilité de 1 quand le nombre de variables tend vers l'infini.

La réduction de 3-SAT au problème de l'existence d'un siphon minimal traité dans [23] est obtenue avec des réseaux de Petri dont la structure est illustrée dans la figure 6. Il est important de noter que dans ce codage, le réseau de Petri a un degré entrant maximum (pour q_0) linéaire par rapport au nombre de clauses et un degré maximal sortant (pour t_0) linéaire par rapport au nombre de variables.

Sans surprise, cette famille de réseaux de Petri fournit un banc d'essai difficile pour l'énumération des siphons minimaux. La Table 3 contient les résultats expérimentaux de ces réseaux de Petri générés [1]. Nous considérons un time-out de deux secondes, le symbole "-" signifie que le délai de deux secondes n'a pas suffit pour énumérer tous les siphons minimaux. Les résultats de cette section sont obtenus en utilisant un PC avec un processeur intel Core2 Quad 2.8 GHz et 8 Go de mémoire. Cette table contient les informations qui concernent le codage 3-SAT et le réseau de Petri correspondant : pour chaque 3-SAT, nous fournissons le nombre de variables booléennes, le nombre de clauses et le ratio α, pour le réseau de Petri correspondant nous donnons le nombre de places, le nombre de transitions et la densité (ratio du nombre de transitions sur le nombre de places). Le temps de calcul de l'énumération de tous les siphons minimaux en fonction de α est représenté dans la figure 7 ; il est visible que le temps de calcul reste exponentiel autour de la valeur critique 4.26 de α.

De même, des réseaux de Petri générés aléatoirement avec des degrés linéaires par rapport au nombre de sommets (places et transitions) représentent aussi des instances difficiles. Par contre, des réseaux de Petri aléatoires ayant des degrés de l'ordre de 10 comme c'est le cas des modèles biochimiques sont des instances faciles pour le problème d'énumération des siphons minimaux. Ainsi, bien que les modèles de réactions biochimiques soient de grande taille, en terme de nombre de places, leurs degré reste borné à une petite valeur ce qui explique pourquoi le problème d'énumération des siphons minimaux est si facile en pratique.

7 Conclusion

Les siphons et les pièges définissent des groupement significatifs de composants qui exposent un comportement particulier durant l'évolution dynamique d'un système biochimique quelles que soient les valeurs des paramètres du modèle.

Nous avons décrit un modèle booléen pour ce problème et nous avons comparé deux méthodes pour satisfaire ces contraintes de façon à énumérer l'ensemble des siphons et des pièges minimaux.

Le solveur SAT est le plus efficace sur les réseaux de grande taille. Le programme PLC(B) est néanmoins

1. Ce banc d'essai est disponible sur ce lien.

modèle	# siphons	# places	# transitions	algorithme dédié	sat	GNU Prolog
carte de Kohn	81	509	775	28	1	221
BIOMD000000175	3042	118	194	∞	137000	∞
BIOMD000000205	32	194	313	21	1	34
BIOMD000000239	64	51	72	2980	1	22
1454	60	39	48	15	1	11
1479	168	58	78	47	4	220
1516	1133	68	102	2072	4739	163 248

TABLE 1 – Performance sur les instances les plus difficiles.

Base de données	# modèles	# siphons min-max (avg.)	taille des siphons min-max (avg.)	temps total		
				algorithme dédié	SAT	GNU Prolog
Biomodels.net	403	0-64 (4.21)	1-413 (3.10)	19734	611	231
Petriweb	79	0-168 (6.24)	1-24 (3.36)	2757	457	240

TABLE 2 – Performance sur l'ensemble du banc d'essai.

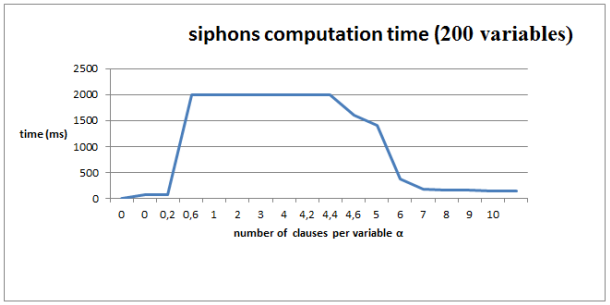


FIGURE 7 – le temps de l'énumération de tous les siphons minimaux avec un time-out de 2 secondes établi une transition de phase au voisinage de la valeur de 4.26 du ratio du nombre de clauses par le nombre de variables.

plus efficace d'au moins un ordre de grandeur que l'algorithme dédié de l'état de l'art [5].

Les deux méthodes sont capables de résoudre tous les problèmes des dépôts de Petriweb et de Biomodels.net (daté mars 2012) dans un temps court ce qui démontre l'applicabilité de cette approche. Notons aussi que le modèle PLC(B) pour calculer les siphons et les pièges minimaux est une extension du modèle PLC(DF) pour la recherche des P- et T-invariants [20].

L'efficacité étonnante de ces méthodes appliquées aux modèles biochimiques de biomodels.net a été expliquée en montrant que le degré des réseaux de Petri est borné dans les modèles biologiques.

L'idée d'appliquer les solveurs SAT ou la programmation logique par contraintes aux problèmes classiques de la communauté des réseaux de Petri n'est pas nouvelle, mais ces approches ont jusque-là été davantage appliquées à la vérification de modèles. Nous sommes persuadés que les problèmes structurels peuvent également bénéficier du savoir-faire développé dans la communauté de la modélisation booléenne. Cela semble être particulièrement intéressant pour la communauté de la biologie des systèmes, où les progrès technologiques récents nécessitent de plus en plus d'outils d'analyse puissants et d'expertise pour des problèmes d'optimisation.

Références

[1] Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos, François Fages, and Vincent Schächter. Modeling and querying biochemical interaction networks. *Theoretical Computer Science*, 325(1) :25–44, September 2004.

[2] Feng Chu and Xiao-Lan Xie. Deadlock analysis of petri nets using siphons and mathematical programming. *IEEE Transactions on Robotics and Automation*, 13(6) :793–804, 1997.

[3] R. Cordone, L. Ferrarini, and L. Piroddi. Characterization of minimal and basis siphons with predicate logic and binary programming. In *Proceedings of IEEE International Symposium on Computer-Aided Control System Design*, pages 193–198, 2002.

[4] R. Cordone, L. Ferrarini, and L. Piroddi. Some results on the computation of minimal siphons

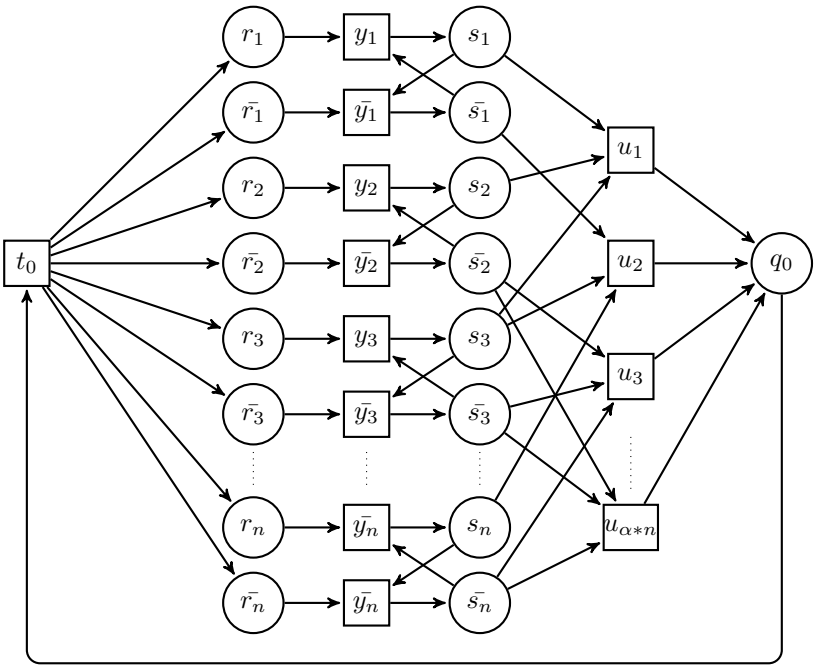


FIGURE 6 – Réseaux de Petri de la réduction 3-SAT

in petri nets. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, Maui, Hawaii USA, dec 2003.

[5] Roberto Cordone, Luca Ferrarini, and Luigi Piroddi. Enumeration algorithms for minimal siphons in petri nets based on place constraints. *IEEE transactions on systems, man and cybernetics. Part A, Systems and humans*, 35(6) :844–854, 2005.

[6] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 21–27. AAAI press, 1993.

[7] Daniel Diaz and Philipe Codognet. Design and implementation of the GNU Prolog system. *Journal of Functional and Logic Programming*, 6, October 2001.

[8] R. Goud, Kees van Hee, R. Post, and J. van der Werf. Petriweb : A repository for petri nets. In Susanna Donatelli and P. Thiagarajan, editors, *Petri Nets and Other Models of Concurrency - ICATPN 2006*, volume 4024 of *Lecture Notes in Computer Science*, pages 411–420. Springer-Verlag, 2006.

[9] Monika Heiner, David Gilbert, and Robin Donaldson. Petri nets for systems and synthetic biology. In M. Bernardo, P. Degano, and G. Zavattaro, editors, *8th Int. School on Formal Methods for the Design of Computer, Communication and Software Systems : Computational Systems Biology SFM'08*, volume 5016 of *Lecture Notes in Computer Science*, pages 215–264, Bertinoro, Italy, February 2008. Springer-Verlag.

[10] M. Kinuyama and T. Murata. Generating siphons and traps by petri net representation of logic equations. In *Proceedings of 2th Conference of the Net Theory SIG-IECE*, pages 93–100, 1986.

[11] Kurt W. Kohn. Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Molecular Biology of the Cell*, 10(8) :2703–2734, August 1999.

[12] K. Lautenbach. Linear algebraic calculation of deadlocks and traps. In Genrich Voss and Rozenberg, editors, *Concurrency and Nets Advances in Petri Nets*, pages 315–336, New York, 1987. Springer-Verlag.

[13] Nicolas le Novère, Benjamin Bornstein, Alexander Broicher, Mélanie Courtot, Marco Donizelli, Harish Dharuri, Lu Li, Herbert Sauro, Maria Schilstra, Bruce Shapiro, Jacky L. Snoep, and Michael Hucka. BioModels Database : a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acid Research*, 1(34) :D689–D691, January 2006.

model	#	Petri net view			3-SAT view			time
	siphons	# places	# transitions	density	# variables	# clauses	α	(ms)
pn0.xml	201	801	401	0,5	200	0	0	79
pn0.0.xml	201	801	401	0,5	200	0	0	79
pn0.2.xml	-	801	441	0,56	200	40	0,2	2000
pn0.6.xml	-	801	521	0,65	200	120	0,6	2000
pn1.xml	-	801	601	0,751	200	200	1	2000
pn2.xml	-	801	801	1	200	400	2	2000
pn3.xml	-	801	1001	1,24	200	600	3	2000
pn4.xml	-	801	1201	1,49	200	800	4	2000
pn4.2.xml	-	801	1241	1,54	200	840	4,2	2000
pn4.4.xml	200	801	1281	1,59	200	880	4,4	1596
pn4.6.xml	200	801	1321	1,64	200	920	4,6	1411
pn5.xml	200	801	1401	1,74	200	1000	5	370
pn6.xml	200	801	1601	1,99	200	1200	6	175
pn7.xml	200	801	1801	2,24	200	1400	7	157
pn8.xml	200	801	2001	2,49	200	1600	8	157
pn9.xml	200	801	2201	2,74	200	1800	9	133
pn10.xml	200	801	2401	2,99	200	2000	10	137

TABLE 3 – Performance sur le banc d'essai généré.

[14] M. Minoux and K. Barkaoui. Deadlocks and traps in petri nets as horn-satisfiability solutions and some related polynomially solvable problems. *Discrete Applied Mathematics*, 29 :195–210, 1990.

[15] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of sat problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 459–465. AAAI press, 1992.

[16] Tadao Murata. Petri nets : properties, analysis and applications. *Proceedings of the IEEE*, 77(4) :541–579, April 1989.

[17] Faten Nabli. Finding minimal siphons as a csp. In *CP'11 : The Seventeenth International Conference on Principles and Practice of Constraint Programming, Doctoral Program*, pages 67–72, September 2011.

[18] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, New Jersey, 1981.

[19] Venkatramana N. Reddy, Michael L. Mavrovouniotis, and Michael N. Liebman. Petri net representations in metabolic pathways. In Lawrence Hunter, David B. Searls, and Jude W. Shavlik, editors, *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 328–336. AAAI Press, 1993.

[20] Sylvain Soliman. Finding minimal P/T-invariants as a CSP. In *Proceedings of the fourth Workshop on Constraint Based Methods for Bioinformatics WCB'08, associated to CPAIOR'08*, May 2008.

[21] Lubert Stryer. *Biochemistry*. Freeman, New York, 1995.

[22] S. Tanimoto, M. Yamauchi, and T. Watanabe. Finding minimal siphons in general petri nets. *IEICE Trans. on Fundamentals in Electronics, Communications and Computer Science*, pages 1817–1824, 1996.

[23] M. Yamauchi and T. Watanabe. Time complexity analysis of the minimal siphon extraction problem of petri nets. *EICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, pages 2558–2565, 1999.

[24] Ionela Zevedei-Oancea and Stefan Schuster. Topological analysis of metabolic networks based on petri net theory. *In Silico Biology*, 3(29), 2003.

Compilation de CSP en Set-labeled Diagram

Alexandre Niveau * **Hélène Fargier** † **Cédric Pralet** ‡

* CRIL, Lens
† IRIT/RPDMP, Toulouse
‡ ONERA/DCSD, Toulouse
niveau@cril.fr fargier@irit.fr cpralet@onera.fr

Résumé

La plupart des requêtes associées aux CSPs sont NP-difficiles, et doivent pourtant parfois être exécutées en ligne et en temps limité. Dans ce cas, la résolution du CSP n'est pas assez efficace, voire impossible. Des structures ont été proposées, tels les MDDs, pour *compiler* les CSPs et rendre leur exploitation en ligne efficace. Les MDDs sont des DAGs orientés dont chaque nœud représente l'assignation d'une variable ; l'ensemble des solutions d'un CSP correspond à l'ensemble des chemins du MDD correspondant. Dans cet article, nous étudions la relaxation de deux restrictions usuellement imposées aux MDDs, l'ordonnancement et la non-répétition des variables, introduisant une structure de compilation nommée « set-labeled diagrams » (SDs). Un CSP peut être compilé en SD en suivant la trace de l'arbre de recherche exploré par un solveur de CSP. L'impact des restrictions susnommées peut être étudié en faisant varier les heuristiques utilisées par le solveur lors de la résolution.

1 Introduction

Les problèmes de satisfaction de contraintes (CSPs) fournissent un cadre puissant permettant de représenter une grande variété de problèmes, tels des problèmes de planification ou de configuration. On peut considérer divers types de requêtes sur un CSP, comme l'extraction d'une solution (la requête la plus classique), la cohérence forte des domaines, l'ajout ou le retrait de nouvelles contraintes (CSP dynamique), voire une combinaison de plusieurs de ces requêtes. Par exemple, la résolution interactive d'un problème de configuration revient à ajouter et retirer des contraintes (unaires) tout en maintenant la cohérence forte des domaines — c'est-à-dire que chaque valeur d'un domaine doit faire partie d'au moins une solution).

La plupart des requêtes sont NP-difficiles ; elles doivent cependant être parfois effectuées en ligne. Une façon possible de résoudre cette contradiction consiste à compiler l'ensemble des solutions d'un CSP en un *diagramme de décision multivalué* (MDD, *multivalued decision diagram*), c'est-à-dire un graphe dont les nœuds sont étiquetés par des variables et les arcs représentent des valeurs assignées à ces variables. Dans de tels diagrammes, chaque chemin de la racine au puits représente une solution du CSP. Un certain nombre d'opérations, notamment les requêtes précédemment citées, sont faisables sur un MDD en temps polynomial en sa taille. Théoriquement, cette taille peut être exponentiellement plus grande que celle du CSP original, mais en pratique elle reste raisonnable dans un certain nombre d'applications. En effet, leur nature de graphe permet aux MDDs de tirer parti de l'interchangeabilité des valeurs, en gagnant de la place grâce à la fusion de sous-problèmes équivalents. Les diagrammes de décision ont été utilisés dans divers contextes, notamment la configuration de produit [2], les systèmes de recommandation [8], ou encore, dans leur forme booléenne originale, dans la planification [10, 13] et le diagnostic [24].

À notre connaissance, ces travaux ne considèrent que des graphes *ordonnés*, c'est-à-dire que l'ordre dans lequel les variables sont rencontrées le long d'un chemin est fixé (x ne peut apparaître avant y dans un chemin, et après y dans un autre), et « *read-once* », c'est-à-dire que les variables ne peuvent être répétées le long d'un chemin. Ce n'est cependant pas une nécessité pour bien des applications ; dans le cas booléen, [5] ont montré que les OBDDs pouvaient être avantageusement remplacés par des FBDDs (*free BDDs*), qui sont « *read-once* » mais pas ordonnés. Dans cet article, nous utilisons le langage des « diagrammes à étiquettes ensemblistes » (SDs, *set-labeled diagrams*) [19], qui généralise les MDDs en relâchant ces deux

contraintes à la fois.

Il est important de noter que nous ne nous intéressons pas à l'utilisation de MDDs pour représenter une contrainte à la fois, avec l'objectif d'utiliser les formes compilées dans un solveur de CSP. Dans ce cas, les requêtes seraient principalement de la propagation. Notre objectif est différent : nous voulons compiler l'ensemble des solutions d'un CSP complet, de manière à pouvoir exécuter en ligne diverses requêtes sur la forme compilée.

Dans cet article, nous étudions une méthode de compilation en SDs, qui consiste à appliquer le concept de « DPLL *with a trace* » [14] à un solveur de CSP. Nous présentons un compilateur générique, permettant de construire des SDs tout en bénéficiant des techniques de programmation par contraintes disponibles dans les solveurs. Avec ce compilateur, relâcher les hypothèses d'ordonnancement statique ou de « *read-once* » dépend des heuristiques de branchement et de choix de variables utilisées pendant la recherche par le solveur. Nous étudions diverses heuristiques possibles et présentons des résultats expérimentaux, obtenus grâce à l'implantation de notre algorithme sur le solveur Choco [9].

Le papier est organisé comme suit : nous présentons tout d'abord dans la section 2 le cadre formel des SDs, qui généralisent les MDDs. Puis, dans la section 3, nous décrivons notre algorithme de compilation. La section 4 détaille les heuristiques que nous avons utilisées, et les résultats de nos expérimentations.

2 Set-labeled Diagrams

2.1 Structure et sémantique

Nous donnons tout d'abord une définition générale des *set-labeled diagrams*, puis nous nous restreignons à un cadre spécifique.

Définition 1 (Set-labeled diagram)**.** Soit $\mathcal{V}$ un ensemble de variables et $\mathcal{E}$ un ensemble d'ensembles. Un *diagramme à étiquettes ensemblistes* (*set-labeled diagram*, SD) est un graphe acyclique orienté ayant au plus une racine et au plus une feuille (appelée le *puits*). Chaque nœud (excepté le puits) est étiqueté par une variable de $\mathcal{V}$ ou par le symbole disjonctif $\veebar$. Chaque arc est étiqueté par un ensemble, élément de $\mathcal{E}$.

Cette définition implique peu de restrictions sur les étiquettes des nœuds et des arcs. Les SDs généralisent ainsi un certain nombre de structures représentant des ensembles de solutions, telles les BDDs [7] (*binary decision diagrams*, en prenant des variables booléennes et $\mathcal{E} = \{\bot, \top\}$), les MDDs [22, 25, 3, 4] (*multivalued decision diagrams*, en prenant des variables discrètes et $\mathcal{E}$ un ensemble de singletons), et les automates à intervalles [20] et *interval diagrams* [23] (en prenant $\mathcal{E}$ un ensemble d'intervalles).

Dans la suite, nous nous restreignons à un cadre proche de celui des MDDs : les domaines des variables sont des parties finies de $\mathbb{Z}$. En revanche, contrairement aux MDDs, les arcs ne sont pas étiquetés par des singletons mais par des ensembles finis de $\mathbb{Z}$. Dans cet article, nous considérerons des ensembles explicitement énumérés — utiliser des ensembles ne fait pas gagner d'espace dans ce cas (l'intérêt est de pouvoir définir une nouvelle restriction structurelle, la convergence). Néanmoins, les SDs sont plus généraux que les MDDs, notamment car l'ordre des variables et leur répétition le long d'un chemin ne sont pas contraints, mais aussi car ils peuvent être *non-déterministes* : les ensembles étiquetant les arcs sortant d'un même nœud ne sont pas forcément deux à deux disjoints. Les SDs déterministes sont appelés dSDs.

Notons qu'un SD peut être vide (n'avoir aucun nœud) ou ne contenir qu'un seul nœud (qui est alors à la fois racine et puits). La figure 1 donne un exemple de SD.

Nous utilisons les notations suivantes : le domaine d'un variable $x \in \mathcal{V}$ est noté $\mathrm{Dom}(x)$. Par convention, on note $\mathrm{Dom}(\veebar) = \{0\}$. On suppose que $\mathcal{V}$ est totalement ordonné, et dans un ensemble noté $X = \{x_1, \ldots, x_k\} \subseteq \mathcal{V}$, les variables soient ordonnés dans l'ordre croissant de leur indice. On note alors $\mathrm{Dom}(X) = \mathrm{Dom}(x_1) \times \cdots \times \mathrm{Dom}(x_k)$, et $\vec{x}$ représente une X-instanciation des variables de X, c'est-à-dire que $\vec{x} \in \mathrm{Dom}(X)$. Enfin, la valeur assignée à x_i dans l'instanciation $\vec{x}$ est notée $\vec{x}_{|x_i}$ (par convention, $\vec{x}_{|\veebar} = 0$ pour tout X). Le cardinal d'un ensemble S est noté $|S|$.

Soit φ un SD, N un nœud et E un arc de φ. On note :

- $\mathrm{Var}(\varphi)$ l'ensemble des variables mentionnées dans φ ;
- $\mathrm{Root}(\varphi)$ la racine et $\mathrm{Sink}(\varphi)$ le puits de φ ;
- $\|\varphi\|$ la *taille* de φ, c'est-à-dire la somme des cardinalités des ensembles et des domaines des variables mentionnés dans φ ;
- $\mathrm{Out}(N)$ (resp. $\mathrm{In}(N)$) l'ensemble des arcs sortants (resp. entrants) du nœud N ;
- $\mathrm{Var}(N)$ la variable étiquetant N (par convention $\mathrm{Var}(\mathrm{Sink}(\varphi)) = \veebar$) ;
- $\mathrm{Src}(E)$ le nœud duquel sort l'arc E et $\mathrm{Dest}(E)$ le nœud dans lequel il entre ;
- $\mathrm{Lbl}(E)$ l'ensemble étiquetant l'arc E ;
- $\mathrm{Var}(E) = \mathrm{Var}(\mathrm{Src}(E))$ la variable relative à E.

Un SD est une représentation compacte d'une fonction booléenne sur des variables discrètes. Cette fonc-

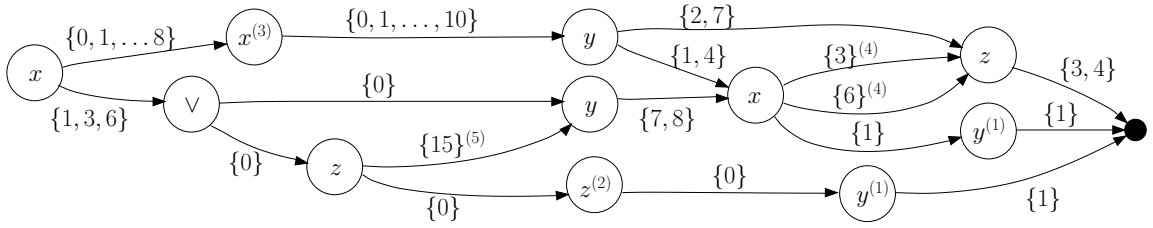


FIGURE 1 – Un exemple de SD. Les variables ont toutes pour domaine l'ensemble $\{0, 1, \ldots, 10\}$. Ce SD n'est pas réduit (voir section 2.2) : les deux nœuds marqués $^{(1)}$ sont isomorphes ; le nœud $^{(2)}$ est bégayant ; le nœud $^{(3)}$ est non-décisif ; les arcs marqués $^{(4)}$ sont contigus ; l'arc $^{(5)}$ est mort.

tion est appelée *interprétation* du SD.

Définition 2 (Sémantique d'un SD)**.** Soit φ un SD, et $X = \mathrm{Var}(\varphi)$. L'*interprétation* de φ est la fonction $[\![\varphi]\!]$, de $\mathrm{Dom}(X)$ vers $\{\top, \bot\}$, et définie comme suit : pour toute X-instanciation $\vec{x}$, $[\![\varphi]\!](\vec{x}) = \top$ si et seulement s'il existe un chemin p de la racine au puits de φ, tel que pour tout arc E de p, $\vec{x}_{|\,\mathrm{Var}(E)} \in \mathrm{Lbl}(E)$.

Une X-instanciation $\vec{x}$ est *modèle* de φ si et seulement si elle vérifie $[\![\varphi]\!](\vec{x}) = \top$. On note $\mathrm{Mod}(\varphi)$ l'ensemble des modèles de φ.

φ est *équivalent* à un autre SD ψ (ce que l'on note $\varphi \equiv \psi$) ssi $\mathrm{Mod}(\varphi) = \mathrm{Mod}(\psi)$.

L'interprétation du SD vide renvoie toujours $\bot$, puisqu'il ne contient aucun chemin de la racine au puits. Inversement, l'interprétation du SD-feuille renvoie toujours $\top$; en effet, le seul chemin de la racine au puits de ce SD ne contient aucun arc.

Définition 3 (Cohérence, validité, contexte)**.** Soit φ un SD, et $X = \mathrm{Var}(\varphi)$.

φ est dit *cohérent* (resp. *valide*) si et seulement si $\mathrm{Mod}(\varphi) \neq \varnothing$ (resp. $\mathrm{Mod}(\varphi) = \mathrm{Dom}(X)$).

Un entier $\omega \in \mathbb{Z}$ est dit *cohérent* pour une variable y dans φ si et seulement s'il existe un modèle $\vec{x}$ de φ tel que $\vec{x}_{|y} = \omega$.

L'ensemble des valeurs cohérentes de y dans φ est appelé *contexte* de y dans φ, et noté $\mathrm{Ctxt}_\varphi(y)$.

Contrairement au cas des MDDs, décider si un SD est cohérent ou non est un problème difficile. Une des raisons est que les différents ensembles qui restreignent le domaine d'une même variable le long d'un chemin peuvent être disjoints, ce qui implique que ledit chemin ne correspond à aucun modèle. Il faudrait donc potentiellement parcourir tous les chemins d'un SD pour décider de sa cohérence. Pour éviter cela, une possibilité est de considérer des SDs dont les ensembles relatifs à une même variable ne peuvent que rétrécir le long d'un chemin. Nous appelons cette propriété *convergence* ; elle est illustrée figure 2. La convergence généralise la propriété de « *read-once* » définie dans le contexte des « free BDDs » ; en effet, si aucune variable n'est répétée, le SD est trivialement convergent.

Définition 4 (SD convergent et *read-once*)**.** Un arc E d'un SD φ est *convergent* si et seulement si tout arc E' sur un chemin de la racine de φ à $\mathrm{Src}(E)$ tel que $\mathrm{Var}(E) = \mathrm{Var}(E')$ vérifie $\mathrm{Lbl}(E) \subseteq \mathrm{Lbl}(E')$.

Un SD *convergent* (*focusing SD*, FSD) est un SD ne contenant que des arcs convergents.

Un SD *read-once* (RSD) est un SD dans lequel il n'existe aucun chemin contenant deux nœuds étiquetés par une même variable.

On note dFSD (resp. dRSD) un SD à la fois déterministe et convergent (resp. *read-once*). Enfin, il est possible d'imposer un ordre sur les variables, et retrouver les MDDs dans leur définition usuelle [1] [22, 25, 3, 4].

Définition 5 (SD ordonné)**.** Soit X un ensemble de variables et $<$ un ordre total sur X. Un SD est *ordonné selon* $<$ si et seulement si tout couple de nœuds (N, M) tel que N est ancêtre de M vérifie $\mathrm{Var}(N) < \mathrm{Var}(M)$.

Un SD déterministe et ordonné selon $<$ est appelé un $\mathrm{MDD}_<$. Le langage [2] `MDD` est l'union des langages $\mathtt{MDD}_<$ pour tout $<$.

Nous montrons à présent comment réduire un SD, de façon à le rendre aussi compact que possible, et gagner de l'espace mémoire.

2.2 Réduction

Comme un BDD, un SD peut être réduit en taille sans que sa sémantique ne change. La réduction est basée sur plusieurs opérations ; certaines sont des généralisations directes de celles introduites dans le contexte des BDDs [7], telles la fusion des nœuds isomorphes

1. La définition originale des MDDs n'exige ni déterminisme, ni ordonnancement. Néanmoins, les travaux utilisant cette structure se restreignant systématiquement à des graphes ordonnés et déterministes, nous décidons arbitrairement de nommer « MDDs » les SDs ordonnés et déterministes.

2. Un *langage* est un ensemble de graphes muni d'une fonction d'interprétation. On note `SD` le langage des SDs, `FSD` le langage des FSDs, etc.

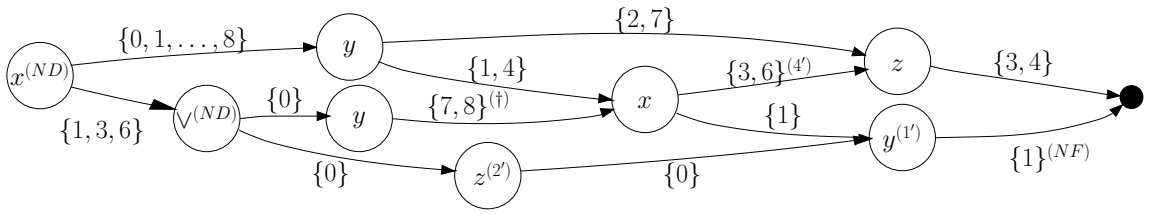


FIGURE 2 – Dans ce SD, tous les arcs sont convergents, excepté celui marqué $^{(NF)}$ (son étiquette n'est pas incluse dans celle de l'arc marqué $^{(\dagger)}$), et tous les nœuds sont déterministes excepté ceux marqués $^{(ND)}$. Ce SD est la forme réduite de celui présenté figure 1 : les nœuds isomorphes, marqués $^{(1)}$, ont été fusionnés en un seul nœud $^{(1')}$; le nœud bégayant $^{(2)}$ a fusionné avec le nœud $^{(2')}$; les arcs contigus, marqués $^{(4)}$, ont fusionné en un seul arc $^{(4')}$; et le nœud non-décisif $^{(3)}$ et l'arc mort $^{(5)}$ ont été supprimés.

(qui sont équivalents) et non-décisifs (qui sont automatiquement franchis), tandis que d'autres sont spécifiques aux SDs, comme la suppression des arcs morts (qui ne sont jamais franchis), et la fusion des arcs contigus (qui ont la même source et la même destination) et des nœuds bégayants (des décisions successives relatives à une même variable). Toutes ces notions sont illustrées sur le SD de la figure 1, et la forme réduite de ce SD est présentée figure 2.

Définition 6.

- Deux arcs E_1 et E_2 sont *contigus* ssi $\mathrm{Src}(E_1) = \mathrm{Src}(E_2)$ et $\mathrm{Dest}(E_1) = \mathrm{Dest}(E_2)$.
- Deux nœuds N_1 et N_2 sont *isomorphes* ssi $\mathrm{Var}(N_1) = \mathrm{Var}(N_2)$ et il existe une bijection σ de $\mathrm{Out}(N_1)$ vers $\mathrm{Out}(N_2)$ vérifiant $\forall E \in \mathrm{Out}(N_1), \mathrm{Lbl}(E) = \mathrm{Lbl}(\sigma(E))$ et $\mathrm{Dest}(E) = \mathrm{Dest}(\sigma(E))$.
- Un arc E est *mort* ssi $\mathrm{Lbl}(E) \cap \mathrm{Dom}(\mathrm{Var}(E)) = \varnothing$.
- Un nœud N est *non-décisif* ssi $|\mathrm{Out}(N)| = 1$ et $E \in \mathrm{Out}(N)$ vérifie $\mathrm{Dom}(\mathrm{Var}(E)) \subseteq \mathrm{Lbl}(E)$.
- Un nœud non-racine N est *bégayant* ssi tous ses parents sont étiquetés par $\mathrm{Var}(N)$, et soit $\sum_{E \in \mathrm{Out}(N)} |E| = 1$, soit $\sum_{E \in \mathrm{In}(N)} |E| = 1$.

Définition 7 (Forme réduite)**.** Un SD φ est dit *réduit* ssi aucun nœud de φ n'est isomorphe à un autre, bégayant ou non-décisif, et aucun arc de φ n'est mort ou contigu à un autre.

La réduction peut être effectuée en temps polynomial en la taille du graphe.

Proposition 8 (Réduction)**.** *Soit* L *un sous-langage de* SD *parmi* $\{\mathtt{SD}, \mathtt{FSD}, \mathtt{dSD}, \mathtt{dFSD}, \mathtt{MDD}, \mathtt{MDD}_<\}$. *Il existe un algorithme polynomial qui associe tout* φ *de* L *à un SD réduit* φ' *de* L *équivalent à* φ *et vérifiant* $|\varphi'| \leq |\varphi|$.

3 Algorithme de compilation : « Choco with a Trace »

3.1 État de l'art

La compilation de connaissances est un domaine principalement étudié du point de vue théorique. Quelques compilateurs ont été implantés, principalement dans le cadre booléen — ils permettent de compiler des fonctions à valeur booléennes sur des variables booléennes — notamment les paquetages pour manipuler des OBDDs (tels Buddy [17] et CUDD [21]) et le plus récent « DPLL with a trace » proposé par Huang et Darwiche [14]. La première catégorie de paquetages compile des formules élémentaires (ou des contraintes élémentaires) en diagramme de décision (binaire), et combine incrémentalement les résultats obtenus en utilisant des opérations de conjonction (approche *bottom-up*). Il est toujours possible d'utiliser le cadre booléen pour représenter des domaines multivalués, en utilisant un encodage booléens des domaines du CSP [15] ; néanmoins, il a été montré expérimentalement [11] que les MDDs sont souvent plus compacts que les BDDs pour des problèmes réels (notamment de configuration de produit).

L'inconvénient de l'approche *bottom-up* est qu'elle peut générer des graphes intermédiaires, qui utilisent de l'espace et peuvent être exponentiellement plus grands que le graphe final. L'algorithme « DPLL with a trace » [14] utilise une approche différente pour compiler des formules propositionnelles, qui s'avère efficace en pratique : celle de construire des diagrammes de décision (ainsi que des d-DNNFs) en exploitant l'arbre de recherche d'une exécution de l'algorithme DPLL, qui énumère toutes les solutions d'une CNF.

Cette idée a été adaptée [12] pour construire des MDDs approchés (dont l'ensemble de modèles est une approximation de l'ensemble de solutions du CSP d'entrée) en exploitant la trace d'un algorithme en profondeur d'abord. Une technique similaire est utilisée [18] pour construire des AOMDDs (MDDs avec des nœuds

ET) à partir de la trace d'une recherche AND/OR.

Toutes les approches précédentes se basent sur un ordre des variables prédéterminé (dans le cas des AOMDDs, c'est un ordre arborescent). Nous relâchons ici cette hypothèse : le choix de la prochaine variable sur laquelle brancher peut être *dynamique*, en fonction d'une heuristique. Nous présentons ici une description générale de notre algorithme, que nous avons implanté au-dessus du solveur de CSP Choco [9].

3.2 Description générale

Soit $\mathcal{C} = \langle X, C \rangle$ un CSP défini par un ensemble de contraintes C sur un ensemble de variables X.

En étendant les principes de « DPLL with a trace » des domaines booléens aux domaines entiers, nous introduisons des mécanismes permettant de construire des dFSDs en exploitant l'arbre de recherche d'un solveur de CSP. Cette approche est présentée dans l'algorithme 1. Implanté au-dessus du solveur Choco, nous obtenons le compilateur « *Choco with a trace* ». La fonction principale, `SD_builder`$(\mathcal{C}, X_a)$ prend en entrée un CSP $\mathcal{C}$ et l'ensemble courant X_a des variables instanciées dans $\mathcal{C}$. Elle retourne une représentation compilée de l'ensemble des solutions de $\mathcal{C}$ sous forme de dFSD. L'appel initial du compilateur est `SD_builder`$(\mathcal{C}_0, \varnothing)$, avec $\mathcal{C}_0$ le CSP initial que l'on veut compiler.

Algorithm 1 `SD_builder`$(\mathcal{C}, X_a)$: renvoie un SD représentant l'ensemble des solutions du CSP $\mathcal{C}$. X_a est l'ensemble courant des variables assignées.

```
1:  Propagate(C)
2:  k := Compute_key(C, X_a)
3:  if le cache contient une entrée pour la clé k then
4:      return le SD correspondant à k dans le cache
5:  if C est prouvé incohérent then
6:      return le SD vide
7:  if X_a = X (toutes les variables de C sont assignées) then
8:      return le SD-feuille
9:  Ψ := ∅
10: x := Choose_var(C)
11: R := Split(C, x)
12: for all r ∈ R do
13:     X'_a := X_a
14:     if r est réduit à un singleton then
15:         X'_a := X'_a ∪ {x}
16:     soit ψ_r := SD_builder(C_| Dom(x)←r, X'_a)
17:     Ψ := Ψ ∪ {ψ_r}
18: soit N le nœud N := Get_node(x, Ψ)
19: soit φ le graphe de racine N
20: ranger φ à la clé k dans le cache
21: return φ
```

3.2.1 Recherche standard

La partie classique de la fonction `SD_builder`$(\mathcal{C}, X_a)$ est une recherche générique en profondeur d'abord, énumérant l'ensemble des solutions du CSP $\mathcal{C}$. Elle fonctionne comme suit : tout d'abord, la fonction `Propagate` applique des techniques de propagation au CSP d'entrée, de manière à retirer des domaines certaines valeurs incohérentes ; si le CSP obtenu est localement cohérent (aucun domaine n'est vide), alors (i) la fonction `Choose_var` sélectionne une variable x non encore assignée ($x \notin X_a$), (ii) la fonction `Split` partitionne le domaine courant de x en plusieurs sous-ensembles disjoints, et (iii) la fonction principale est appelée successivement sur chacun des sous-problèmes induits par la partition. À ce niveau, l'implantation des fonctions `Propagate`, `Choose_var` and `Split` n'a pas d'importance ; n'importe quelles techniques et heuristiques peuvent être utilisées.

3.2.2 Ajouts pour la compilation

Pour construire un dFSD représentant l'ensemble des solutions du CSP initial, certains ajouts sont nécessaires ; ils sont encadrés dans l'algorithme 1. L'idée de base est de construire une trace de l'arbre de recherche par le biais de divers mécanismes :

- Soit n le nœud courant de l'arbre de recherche ; on note $\mathcal{C}(n)$ le CSP courant associé à ce nœud. Soit x la variable non-assignée choisie au nœud n, et R la partition de $\mathrm{Dom}(x)$ sur laquelle l'algorithme a décidé de brancher (une branche par élément dans la partition). L'idée est que l'exploration associée à chaque sous-domaine $r \in R$ génère un SD ψ_r ; ce SD représente l'ensemble des solutions du sous-problème $\mathcal{C}(n)_{|\,\mathrm{Dom}(x)\leftarrow r}$, obtenu en réduisant le domaine de x à r. Alors, l'ensemble des solutions du CSP $\mathcal{C}(n)$ sur $X \setminus X_a$ est un SD $\varphi(n)$ dont la racine est étiquetée par x et qui contient, pour chaque $r \in R$ tel que ψ_r n'est pas le SD vide, un arc de la racine à ψ_r.
  Dans la fonction `SD_builder`$(\mathcal{C}, X_a)$, le graphe $\varphi(n)$ est obtenu par l'appel `Get_node`$(x, \{\, \psi_r \mid r \in R \,\})$. En particulier, la fonction `Get_node` vérifie si le graphe $\varphi(n)$ existe déjà dans la *table de nœuds uniques*, qui contient tous les nœuds de SD créés au cours de la recherche. Si le nœud demandé est isomorphe à un nœud contenu dans la table, le nœud déjà existant est retourné ; sinon, le nœud est créé et

ajouté à la table.
- Si n est une feuille de l'arbre de recherche, elle correspond soit à une solution soit à un cul-de-sac. Dans le premier cas, l'algorithme renvoie le SD-feuille (l. 8) : toute instanciation est solution du problème courant. Dans le deuxième cas, il retourne le SD vide (l. 6).
- Les deux points précédents suffisent à compiler un dFSD représentant l'ensemble des solutions du CSP initial. Pour diverses raisons expliquées dans la suite, on maintient lors de la recherche un *cache*, permettant d'éviter à des sous-problèmes équivalents d'être ré-explorés. Plus précisément, chaque fois qu'un sous-problème $\mathcal{C}$ est résolu, une clé $k(\mathcal{C})$ est générée, qui dépend des domaines courants des variables. Cette clé est enregistrée avec le SD représentant ce sous-problème (l. 20). Par la suite, avant de résoudre un nouveau sous-problème $\mathcal{C}'$, sa clé $k(\mathcal{C}')$ est calculée. Si cette clé est déjà présente dans le cache (l. 2), on renvoie directement le SD associé (l. 4).

3.2.3 Structure des SDs obtenus

Les SDs renvoyés par l'algorithme 1 satisfont toujours la propriété de convergence. En effet, les domaines des variables sont systématiquement réduits, soit par le partitionnement de leur domaine, soit par la propagation. Les SDs obtenus sont également déterministes, puisque la fonction `Split` renvoie une *partition* — les sous-ensembles du domaines courants sont donc disjoints.

Cependant, en fonction des heuristiques de branchement et de choix de variable utilisées par `Split` et `Choose_var`, les dFSDs obtenus peuvent varier :
- On obtient des dRSDs si `Split` partage le domaine en singletons, c'est-à-dire si l'algorithme énumère toutes les valeurs possibles des variables durant la recherche. Au contraire, une recherche dichotomique (qui partage les domaines en deux) renvoie des dFSDs non « *read-once* ».
- On obtient des MDDs si `Choose_var` suit un ordre statique ; mais utiliser des heuristiques pour guider le choix des variables (par exemple **MinDomain**) résulte le plus souvent en des dFSDs non-ordonnés.

3.2.4 Caching

Soit $\mathcal{C}$ le CSP courant à un nœud de recherche donné n, et X_a l'ensemble des variables assignées dans $\mathcal{C}$. L'exploration au nœud n retourne un SD ne mentionnant que des variables de $X \setminus X_a$, qui représente l'ensemble des solutions sur $X \setminus X_a$ quand les variables de X_a sont assignées à leur seule valeur possible. L'objectif est de définir des clés de hachage telles que chaque clé regroupe le plus possible de paires $(\mathcal{C}, X_a)$ « équivalentes » selon l'ensemble de leurs solutions sur $X \setminus X_a$.

Intuitivement, le sous-problème courant peut être représenté par une clé listant les domaines courants de toutes les variables. Il est possible, pour réduire la taille de la clé, d'utiliser la technique présentée par Lecoutre et al. [16], qui consiste à retirer de la clé les domaines des variables x qui sont assignées ($x \in X_a$) et utilisées uniquement dans des contraintes *nécessairement satisfaites* dans le sous-problème courant (parfois appelées contraintes *universelles*). Il est prouvé [16] que ce mécanisme conserve l'ensemble de solutions.

Soulignons que le cache est utile pour deux raisons :
- si le sous-problème courant est incohérent, l'enregistrer en cache peut permettre d'éviter des calculs redondants, si la recherche mène à un sous-problème équivalent ultérieurement. C'est l'utilisation qu'en font Lecoutre et al. [16], qui appliquent cette idée à des algorithmes dont l'objectif est d'exhiber une solution (et non toutes les solutions).
- si le sous-problème courant est cohérent, en plus d'éviter des calculs, enregistrer le SD associé en cache peut permettre de réduire la taille du SD final, si les heuristiques de sélection de variable et de partage des domaines sont dynamiques. En effet, sans le cache, des CSPs équivalents seraient explorés indépendamment, avec des ordres de variables et des partages de domaines différents, surtout si des heuristiques aléatoires sont utilisées. Cela conduirait à des SDs non-isomorphes bien qu'ayant la même interprétation, ce que les opérations de réduction ne détectent pas. En utilisant le cache, on maximise les chances que les SDs équivalents soient toujours isomorphes, ce qui réduit de fait la taille du SD final.

3.2.5 Minimisation du cache

Enregistrer tous les sous-problèmes rencontrées peut conduire à un cache de très grande taille. Pour garder un cache de taille raisonnable (qui puisse être conservé dans la mémoire vive), nous avons choisi d'utiliser un mécanisme de « *restart* » : la taille du cache est limitée à une valeur arbitraire, et à chaque fois qu'elle est atteinte, une partie des entrées sont supprimées — l'objectif étant de garder les entrées les plus intéressantes.

Le critère de sélection des entrées à supprimer que nous avons choisi est le suivant : celles qui ont été le moins utilisées d'abord, puis les plus anciennes, puis les plus longues (qui correspondent aux plus petits sous-problèmes).

Ce compromis permet à l'algorithme d'être plus rapide et d'utiliser moins de mémoire, et ainsi de compiler des problèmes plus gros. En revanche, il n'améliore pas le SD résultant ; cela peut même être le contraire (voir sous-section précédente).

3.3 Heuristiques

Nous détaillons ici les différentes alternatives que nous avons considérées pour la fonction `Choose_var`.

3.3.1 Heuristiques classiques

Nous avons utilisé des heuristiques standard de sélection de variables pour la résolution de CSPs :

- **MinDomain** La variable choisie est celle qui a le plus petit domaine.
- **Dom/WDeg** La variable choisie est celle qui minimise le ratio $|\mathrm{Dom}(x)|/\deg(x)$, où $\deg(x)$ est le nombre de contraintes sur x (les contraintes étant pondérées en fonction des conflits) [6].
- **Random :** La variable est choisie aléatoirement.

3.3.2 Heuristiques basées sur le graphe de contraintes

Nous avons utilisé des heuristiques basées sur le graphe de contraintes (graphe dont les variables sont des nœuds, qui sont liés par un arc si et seulement s'il existe une contrainte liant les deux variables). Pour une variable x, on note $\mathrm{N}(x)$ l'ensemble des variables liées à x dans le graphe.

Algorithm 2 `Next_var`(O) choisit la variable suivante, en fonction d'un ordre courant $O = \{o_1, \ldots, o_k\}$.

```
1: if O = ∅ then
2:     return Argmax_{x∈X} |N(x)|
3: soit x := Argmax_{x∈X\O} H_S(x)
4: ajouter x à la fin de l'ordre O
5: return x
```

Les heuristiques suivantes sont basées sur l'algorithme 2, et font varier le critère $\mathcal{H}_S(x)$. Elles ont été introduites par Amilhastre [1].

- **HBW** $\mathcal{H}_S(x) = \max_{1 \leq i \leq |O|, o_i \in \mathrm{N}(x)} |O| - i$ (choisit en priorité une voisine de o_1, puis de o_2, etc).
- **HSBW** $\mathcal{H}_S(x) = \sum_{1 \leq i \leq |O|, o_i \in \mathrm{N}(x)} |O| - i$ (choisit une voisine des variables les plus anciennement choisies).
- **MCSInv** $\mathcal{H}_S(x) = |\mathrm{N}(x) \cap O|$ (choisit la variable la plus liée aux variables déjà choisies).

3.3.3 Heuristique exploitant le cache

Nous avons implanté une heuristique visant à maximiser l'utilisation du cache. L'idée est qu'un choix de variable menant à un sous-problème déjà traité limite le nombre de nouveaux nœuds. L'heuristique consiste donc à compter, pour toute variable non-assignée x, le nombre $n_{\mathrm{new}}(x)$ de branchements sur x pour lesquels il sera nécessaire d'ouvrir un nouveau nœud de recherche (autrement dit, ce branchement ne mène ni à un nœud du cache, ni à un problème incohérent). On choisit alors la variable x minimisant $n_{\mathrm{new}}(x)$ (en se rabattant sur **HBW** pour casser les ex-æquo). Nous appelons cette heuristique **MaxHashUse**.

3.3.4 Versions statique et dynamique

Les heuristiques **MinDomain**, **Dom/WDeg** et **MaxHashUse** sont toujours dynamiques : elles génèrent le plus souvent des dFSDs non-ordonnés. En ce qui concerne **HBW**, **HSBW**, **MCSInv** et **Random**, il est possible de choisir un ordre statique avant le début de la recherche, ce qui permet d'obtenir des MDDs, ou de choisir la variable dynamiquement. On appelle **DynHBW**, **DynHSBW**, **DynMCSInv** et **DynRandom** les heuristiques dynamiques correspondantes.

4 Expérimentations

Nous avons considéré les problèmes suivants lors de nos expérimentations avec le compilateur « Choco with a trace » :

- *ObsToMem* est un problème de reconfiguration. Il représente un contrôleur gérant les connexions entre l'outil d'observation et la mémoire de masse d'un satellite.
- *Drone* est un problème de planification, dans lequel un drone doit remplir certains objectifs dans un certain nombre de zones en temps limité.
- *NQueens* est le problème des « n reines ».
- *Star* est le problème consistant à colorier un graphe en étoile (une variable centrale liée à des variables indépendantes les unes des autres).

4.1 Heuristiques de choix de variable

Nous comparons tout d'abord les différentes heuristiques de choix de variable, en fixant `Split` à un partage en singletons — nous obtenons donc des dRSDs. Comme certaines de ces heuristiques n'ont pas de version statique, nous ne comparons que les versions dynamiques. Les résultats sont présentés figure 3. **Random** n'est pas incluse ici pour améliorer la lisibilité des graphiques, car elle donne de trop mauvais résultats ; on peut les trouver dans la sous-section suivante.

Il est intéressant de remarquer que **MinDomain** semble la meilleure heuristique pour *ObsToMem* et *NQueens*, alors qu'elle est bien pire que les autres pour

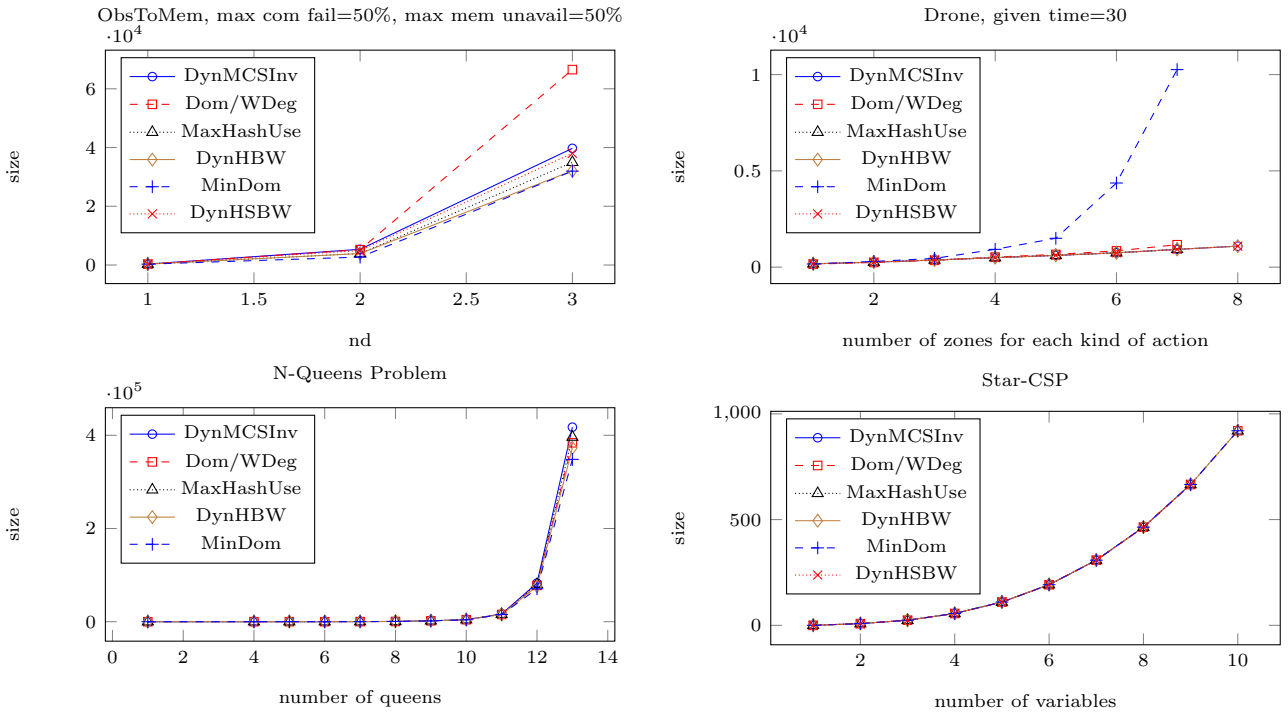


FIGURE 3 – Comparaison entre les heuristiques dynamiques pour les problèmes *ObsToMem*, *Drone*, *NQueens* et *Star*

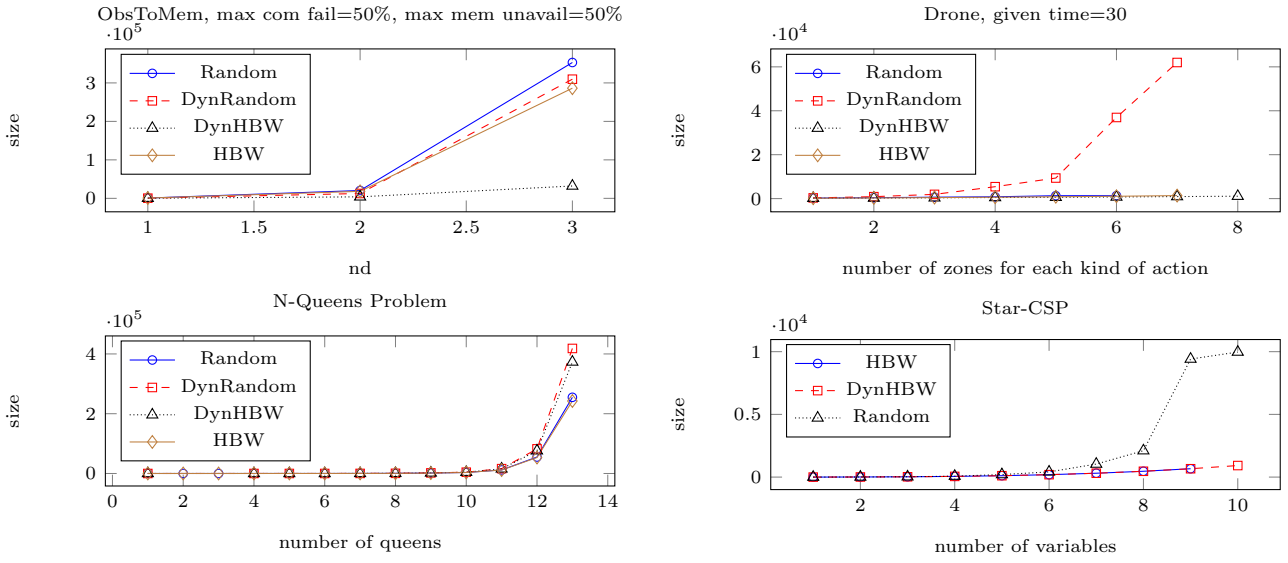


FIGURE 4 – Comparaison entre les versions statique et dynamique de **HBW** et **Random**. Les résultats pour **DynRandom** sur le problème *Star* ne sont pas montrés, étant bien plus mauvais que les autres (la taille du dRSD résultant dépasse les 100 000 pour 7 variables.

Drone. **Dom/WDeg** n'est vraiment efficace dans aucun des trois problèmes. Parmi les heuristiques basées sur le graphe de contraintes, la meilleure est **HBW**. **MaxHashUse** n'est pas mauvaise, mais ne surclasse pas **HBW** ; il semble que regarder seulement une étape en avant ne soit pas suffisant pour maximiser l'utilisation du cache (on choisit la meilleures variable pour le nœud suivant, ouvrant ainsi moins de nouveaux sous-graphes, mais ces sous-graphes sont plus gros). En ce qui concerne *Star*, le fait que toutes les courbes coïncident s'explique en remarquant que toutes les heuristiques choisissent la variable centrale en premier.

4.2 Comparaison entre ordres statique et dynamique

Nous comparons à présent les résultats obtenus en utilisant les versions statique et dynamique d'une même heuristique, avec la même fonction de partage des domaines que dans la sous-section précédente : on obtient respectivement des MDDs et des dRSDs. Les résultats pour **HBW** (meilleure heuristique de la section précédente) et **Random** sont montrés figure 4. On voit que **DynRandom** est beaucoup plus mauvaise que sa version statique ; en effet, le fait d'utiliser un ordre statique augmente fortement la probabilité de tomber sur des nœuds isomorphes. Les résultats pour **DynHBW** sont meilleurs que pour **HBW** statique pour les problèmes réels, mais pas pour les problèmes plus petits (pour *NQueens*, les MDDs obtenus sont même plus petits que les dRSDs). Cependant, on ne peut tirer de conclusions générales sur les qualités relatives des versions statique et dynamique, qui dépendent fortement du problème et de l'heuristique considérés ; sur la figure 5, on voit que pour le problème *Drone*, alors que **DynHBW** et **DynHSBW** coïncident, leurs versions statiques sont tantôt meilleures (**HSBW**) tantôt moins bonnes (**HBW**).

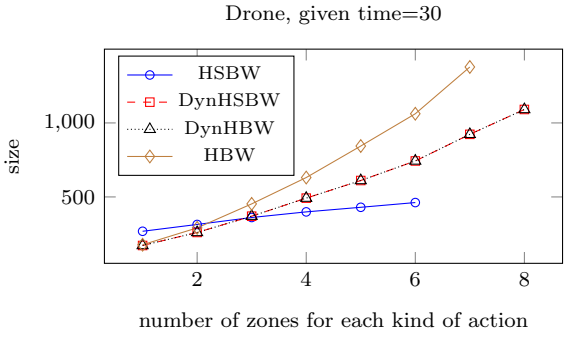


FIGURE 5 – Comparaison entre les versions statique et dynamique de **HBW** et **HSBW** pour le problème *Drone*.

4.3 Influence de la fonction de partage des domaines

Nous étudions à présent comment le choix de la fonction de partage des domaines (`Split`) affecte le SD résultant. Utiliser un partage énumératif (qui divise le domaine en singletons) permet d'obtenir des dRSDs, tandis qu'un partage dichotomique (qui divise le domaine en deux) permet d'obtenir des dFSDs. Les résultats sont présentés figure 6. Ils sont meilleurs dans le premier cas, le nombre de nœuds créés étant bien moindre. Néanmoins, cela n'implique pas qu'un dRSD est toujours plus petit qu'un dFSD équivalent — seulement que notre algorithme n'est apparemment pas efficace pour construire des dFSDs tirant parti du relâchement de la contrainte « *read-once* ».

5 Conclusion

Dans cet article, nous avons introduit les diagrammes à étiquettes ensemblistes, qui généralisent les MDDs en relâchant les propriétés d'ordonnancement, de « *read-once* » et de déterminisme. Nous avons présenté un algorithme permettant de compiler des CSPs discrets en diverses sortes de SDs déterministes (dFSDs, dRSDs et MDDs), en étendant l'approche « DPLL with a trace » sur un solveur de CSP. Nous avons montré comment le choix de certains paramètres de recherche (choix de la variable de branchement et du partage des domaines) affectent la structure du dSD résultant. En utilisant notre implantation de ce compilateur (basée sur le solveur de CSP Choco), sur deux problèmes réels et deux CSPs classiques, nous avons présenté des résultats expérimentaux à propos de l'influence de ces paramètres sur la taille des formes compilées. Ces résultats montrent qu'aucune des heuristiques de choix de variable que nous avons utilisées ne surclasse les autres pour tous les problèmes ; la recherche d'une heuristique globalement efficace reste ouverte. Ils montrent également que le compilateur que nous avons développé semble être utilisable pour compiler des SDs « *read-once* », mais pas purement convergents.

Notre travail s'oriente vers une étude plus approfondie des heuristiques de choix de variable, en particulier **MaxHashUse**. Plus généralement, un de nos objectifs est de compiler efficacement des SDs purement convergents, ainsi que des SDs non-déterministes. Il serait également intéressant d'ajouter des nœuds ET aux SDs, obtenant ainsi un langage proche des AOMDDs ; cela permettrait de comparer la compilation de CSPs en AOMDDs avec des ordres arborescents statiques et dynamiques.

Références

[1] J. Amilhastre. *Représentation par automate d'ensemble de solutions de problèmes de satisfaction de contraintes*. PhD thesis, Université Montpellier II, 1999.

[2] J. Amilhastre, H. Fargier, and P. Marquis. Consistency Restoration and Explanations in Dynamic CSPs — Application to Configuration. *AIJ*, 135(1–2) :199–234, 2002.

[3] J. Amilhastre, P. Vilarem, and M.-C. Vilarem. FA Minimisation Heuristics for a Class of Finite Languages. In *WIA*, pages 1–12, 1999.

[4] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A Constraint Store Based on Multivalued Decision Diagrams. In *CP*, pages 118–132, 2007.

[5] J. Bern, J. Gergov, C. Meinel, and A. Slobodová. Boolean manipulation with free bdd's. first experimental results. In *EDAC-ETC-EUROASIC*, pages 200–207, 1994.

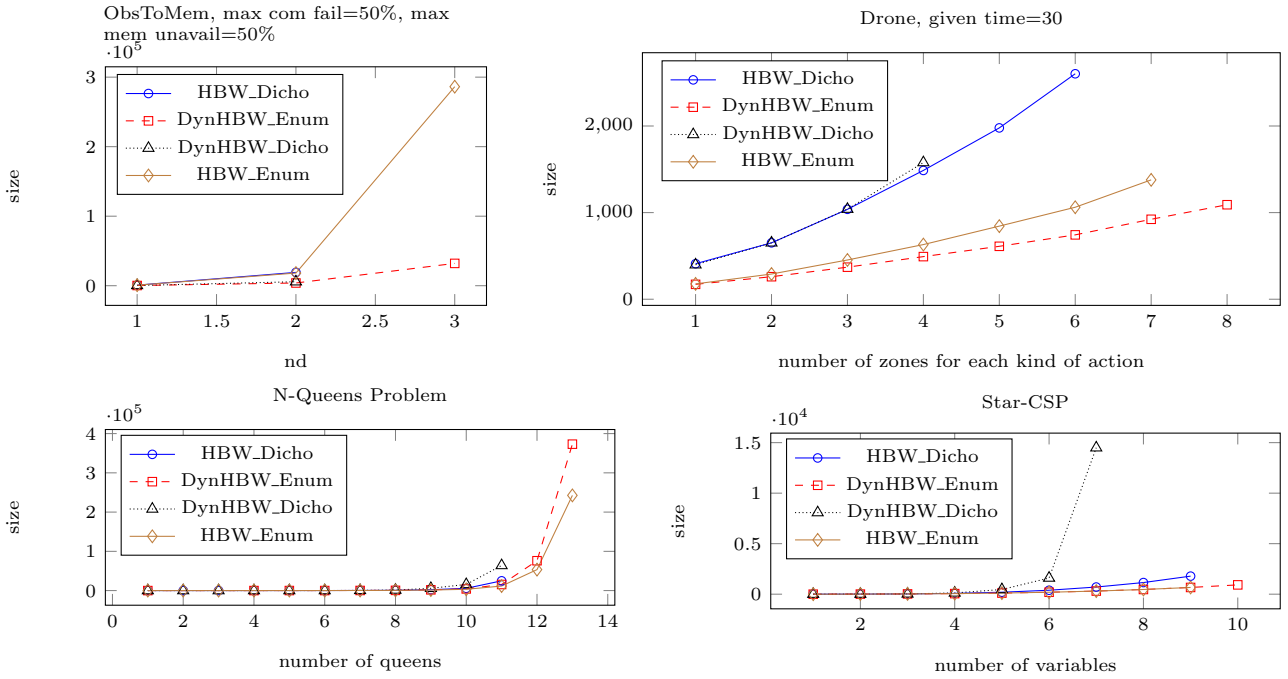


FIGURE 6 – Comparaison entre fonctions de partage de domaines dichotomique et énumérative. La compilation des instances d'ObsToMem non représentées sur le graphique a été interrompue en raison d'un trop grand nombre de nœuds (plus de 700 000).

[6] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *ECAI*, pages 146–150, 2004.

[7] R. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8) :677–691, 1986.

[8] H. Cambazard, T. Hadzic, and B. O'Sullivan. Knowledge Compilation for Itemset Mining. In *ECAI*, pages 1109–1110, 2010.

[9] choco Team. choco : an open source java constraint programming library. Research report 10-02-INFO, Ecole des Mines de Nantes, 2010.

[10] F. Giunchiglia and P. Traverso. Planning as Model Checking. In *ECP*, pages 1–20, 1999.

[11] T. Hadzic, E. Hansen, and B. B. O'Sullivan. On Automata, MDDs and BDDs in Constraint Satisfaction. In *ECAI Workshop on Inference methods based on Graphical Structures of Knowledge (WIGSK)*, 2008.

[12] T. Hadzic, J. N. Hooker, B. O'Sullivan, and P. Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In *CP*, pages 448–462, 2008.

[13] J. Hoey, R. St-Aubin, A. J. Hu, and C. Boutilier. SPUDD : Stochastic Planning Using Decision Diagrams. In *UAI*, pages 279–288, 1999.

[14] J. Huang and A. Darwiche. DPLL with a Trace : From SAT to Knowledge Compilation. In *IJCAI*, pages 156–162, 2005.

[15] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Multi-valued Decision Diagrams : Theory and Applications. *Multiple-Valued Logic*, 4(1–2) :9–62, 1998.

[16] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Transposition tables for constraint satisfaction. In *AAAI*, pages 243–248, 2007.

[17] J. Lind-Nielsen. BuDDy : Binary Decision Diagrams Library Package, release 2.4, 2002. `http://sourceforge.net/projects/buddy/`.

[18] R. Mateescu, R. Dechter, and R. Marinescu. AND/OR multi-valued decision diagrams (AOMDDs) for graphical models. *J. Artif. Intell. Res. (JAIR)*, 33 :465–519, 2008.

[19] A. Niveau, H. Fargier, and C. Pralet. Representing CSPs with set-labeled diagrams : A compilation map. In *Proc. of the 2nd International Workshop on Graph Structures for Knowledge Representation and Reasoning (GKR)*, 2011.

[20] A. Niveau, H. Fargier, C. Pralet, and G. Verfaillie. Knowledge compilation using interval automata and applications to planning. In *ECAI*, pages 459–464, 2010.

[21] F. Somenzi. CUDD : Colorado University Decision Diagram package, release 2.4.1, 2005. `http://vlsi.colorado.edu/~fabio/CUDD/`.

[22] A. Srinivasan, T. Ham, S. Malik, and R. Brayton. Algorithms for discrete function manipulation. In *ICCAD-90*, pages 92 –95, Nov. 1990.

[23] K. Strehl and L. Thiele. Symbolic model checking of process networks using interval diagram techniques. In *Proc. of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 686–692, 1998.

[24] P. Torasso and G. Torta. Model-Based Diagnosis Through OBDD Compilation : A Complexity Analysis. In *Reasoning, Action and Interaction in AI Theories and Systems*, pages 287–305, 2006.

[25] N. R. Vempaty. Solving Constraint Satisfaction Problems Using Finite State Automata. In *AAAI*, pages 453–458, 1992.

Au delà des QCSP pour résoudre des problèmes de contrôle

Cédric Pralet **Gérard Verfaillie**

ONERA – The French Aerospace Lab, F-31055, Toulouse, France
cedric.pralet@onera.fr gerard.verfaillie@onera.fr

Résumé

Les problèmes de satisfaction de contraintes quantifiés (QCSP) sont souvent présentés comme les outils adéquats permettant de modéliser et de résoudre des problèmes de jeu à deux joueurs ou de planification dans l'incertain ou plus généralement des problèmes où l'objectif est de contrôler un système dynamique soumis à des événements incontrôlés. Ce papier montre que, pour de nombreux problèmes de ce type, l'approche standard de type QCSP ou QCSP+ n'est pas la plus appropriée. Les raisons principales en sont que, dans le cadre QCSP/QCSP+, (1) les évolutions possibles du système sont dépliées sur un nombre fixé d'étapes, (2) la notion d'état du système n'est pas explicitement prise en compte et (3) les algorithmes recherchent des stratégies gagnantes à mémoire complète définies comme des arbres de politique plutôt que des stratégies sans mémoire définies comme des fonctions depuis les états vers les décisions. Ce papier propose un nouveau cadre à base de contraintes qui n'a pas ces défauts. Les expérimentations montrent des améliorations de plusieurs ordres de grandeur par rapport à des solveurs de QCSP/QCSP+.

1 Introduction

Les problèmes de satisfaction de contraintes quantifiés (QCSP [3]) ont été introduits pour modéliser et résoudre des problèmes de satisfaction de contraintes (CSP) où la valeur prise par certaines variables est incertaine ou incontrôlable. Formellement, un QCSP est défini par deux éléments : un ensemble de contraintes C et une séquence de quantification $Q = Q_1 x_1 \dots Q_n x_n$ où chaque Q_i est un quantificateur existentiel ou universel ($\exists$ ou $\forall$). Un QCSP défini par $C = \{x_1 + x_3 < x_4, x_2 \neq x_1 - x_3\}$ et $Q = \exists x_1 \forall x_2 \exists x_3 \forall x_4$ doit être interprété comme : "Existe-t-il une valeur de x_1 telle que, pour toute valeur de x_2, il existe une valeur de x_3 telle que, pour toute valeur de x_4, les contraintes de C sont satisfaites ?". Résoudre un QCSP consiste à répondre oui ou non à cette question et, en cas de réponse positive, à produire une stratégie gagnante. Si $\mathbf{d}(x)$ désigne le domaine d'une variable x et A_x l'ensemble des variables quantifiées universellement qui précédent x dans la séquence de quantification, une telle stratégie est généralement définie comme un ensemble de fonctions $f_x : (\prod_{y \in A_x} \mathbf{d}(y)) \rightarrow \mathbf{d}(x)$: une fonction par variable x quantifiée existentiellement. Cet ensemble de fonctions peut être représenté par ce qu'on appelle un arbre de politique. Divers algorithmes ont été proposés ces dernières années pour résoudre un QCSP, depuis les premières techniques utilisant l'arc-cohérence quantifiée binaire ou ternaire (QAC [3, 8]) ou la transformation en formule booléenne quantifiée (QBF [4]) jusqu'aux techniques utilisant la règle dite de valeur pure, l'arc-cohérence quantifiée généralisée [9], le *backjumping* guidé par les conflits [5], la réparation de solutions [13] ou le parcours de droite à gauche de la séquence de quantification [14]. Récemment, une variante de QCSP dénommée QCSP+ [1] a été proposée pour rendre la modélisation plus aisée. L'idée est d'utiliser des séquences de quantification restreinte. Par exemple, une séquence de la forme $\exists x_1 [x_1 \geq 3] \forall x_2 [x_2 \leq x_1] \exists x_3, x_4 [(x_3 \neq x_4) \wedge (x_3 \neq x_1)] C$ doit être interprétée comme : "Existe-t-il une valeur de x_1 telle que $x_1 \geq 3$ et, pour toute valeur de x_2 telle que $x_2 \leq x_1$, il existe des valeurs de x_3 et x_4 telles que $x_3 \neq x_4$, $x_3 \neq x_1$ et toutes les contraintes de C sont satisfaites ?".

QCSP et QCSP+ peuvent être utilisés pour modéliser des problèmes impliquant quelques alternances de quantificateur tels que des problèmes d'ordonnancement dans l'incertain [1]. Ils peuvent aussi être utilisés pour modéliser des problèmes impliquant un grand nombre d'alternances de quantificateur tels que des

problèmes de jeu à deux joueurs ou de planification dans l'incertain. Dans les problèmes de jeu, le but est de déterminer un premier coup pour le joueur 1 tel que, quel que soit le coup du joueur 2, il existe un second coup pour le joueur 1 tel que, quel que soit le coup du joueur 2 ... le joueur 1 gagne. La taille de la séquence de quantification dépend du nombre maximum de coups à considérer. La planification dans l'incertain et plus généralement le problème de contrôle de l'état d'un système dynamique soumis à des événements peuvent être vus comme un jeu contre la nature.

Le but de ce papier est de montrer que, quand l'état du système est complètement observable et quand l'évolution de l'état est markovienne, c'est-à-dire quand l'état courant ne dépend que de l'état et de l'événement précédents et non de tout l'historique des événements, une approche de type QCSP/QCSP+ n'est pas la plus pertinente. Le papier est organisé comme suit. Nous commençons par illustrer pourquoi une approche de type QCSP/QCSP+ n'est pas toujours appropriée (section 2). Nous introduisons ensuite un cadre dénommé MGCSP for *Markovian Game CSP* (section 3) et des algorithmes associés (section 4). Les résultats expérimentaux sont rapportés en section 5. Les preuves sont omises pour des raisons de place.

2 Exemple illustratif

Considérons le jeu *NimFibo*, un *benchmark* QCSP classique. Ce jeu implique deux joueurs A et B qui jouent alternativement. Initialement, N allumettes sont sur la table. Lors du premier coup, le joueur A peut prendre entre 1 et $N-1$ allumettes. Puis, lors de chaque coup, le joueur courant prend au moins une allumette et au plus deux fois le nombre d'allumettes prises par l'autre joueur lors du coup précédent. Le joueur qui prend la dernière allumette gagne. Le problème est de trouver une stratégie gagnante pour le joueur A.

2.1 Approche QCSP/QCSP+

Supposons que N est impair. Pour modéliser ce jeu comme un QCSP+, on peut introduire N variables $r_1, \ldots, r_N$ de domaine $[0..N]$ qui représentent le nombre d'allumettes restantes après chaque coup (N variables parce qu'il y a au plus N coups) et N variables de domaine $[1..N-1]$ qui représentent le nombre d'allumettes prises à chaque coup : a_1, a_3, ..., a_N pour le joueur A et b_2, b_4, ..., b_{N-1} pour le joueur B. Un modèle QCSP+ est alors le suivant :

$$\exists a_1, r_1[r_1 = N - a_1]$$
$$\forall b_2, r_2[b_2 \leq 2a_1, r_2 = r_1 - b_2]$$
$$\exists a_3, r_3[a_3 \leq 2b_2, r_3 = r_2 - a_3]$$
$$\forall b_4, r_4[b_4 \leq 2a_3, r_4 = r_3 - b_4] \ldots$$
$$\exists a_N, r_N[a_N \leq 2b_{N\text{-}1}, r_N = r_{N\text{-}1} - a_N] \ \mathit{True}$$

La figure 1(a) représente une stratégie gagnante exprimée sous la forme d'un arbre de politique pour $N = 15$. Les nœuds circulaires représentent toutes les décisions possibles du joueur B tandis que les nœuds carrés représentent les décisions que doit prendre le joueur A pour gagner.

2.2 Approche à base d'état

L'état du système à chaque étape peut être représenté par trois variables d'état : une variable $j \in \{A, B\}$ représentant le joueur courant, une variable $r \in [0..N]$ représentant le nombre d'allumettes restantes et une variable $p \in [1..N]$ représentant le nombre d'allumettes prises par le joueur précédent. La décision prise lors de chaque tour peut être représentée par deux variables de décision a et b de domaine $[1..N-1]$ représentant le nombre d'allumettes prises respectivement par les joueurs A et B. La décision a est prise avant la décision b. La décision a est sous le contrôle du joueur A alors que la décision b ne l'est pas. Le but est d'atteindre, quelles que soient les décisions du joueur B, un état où $j = B$ (B doit jouer) et $r = 0$ (il ne reste plus d'allumettes). Une solution de ce problème de contrôle peut prendre la forme d'une politique $\pi : \{(A, r, p) \,|\, r \in [1..N], p \in [1..N]\} \rightarrow \mathbf{d}(a)$ qui associe une valeur de a à chaque état dans lequel A doit jouer et il reste au moins une allumette. Il n'est pas nécessaire de définir la politique π pour tous les états, mais uniquement pour ceux qui sont atteignables à partir de l'état initial en suivant π. La figure 1(b) montre une politique solution, tandis que la figure 1(c) représente les états atteignables en suivant cette politique, ainsi que les transitions entre états. Dans la figure 1(b), la deuxième ligne de la première colonne indique par exemple que, dans l'état $(A, 12, 1)$, le joueur A doit prendre une allumette ($a = 1$).

2.3 Comparaison entre les deux approches

Premièrement, l'approche QCSP oblige à borner le nombre d'étapes à considérer et a déplier les évolutions possibles du système sur le nombre maximum d'étapes (ici N). Le nombre de variables à considérer est proportionnel au nombre d'étapes. A l'opposé, l'approche

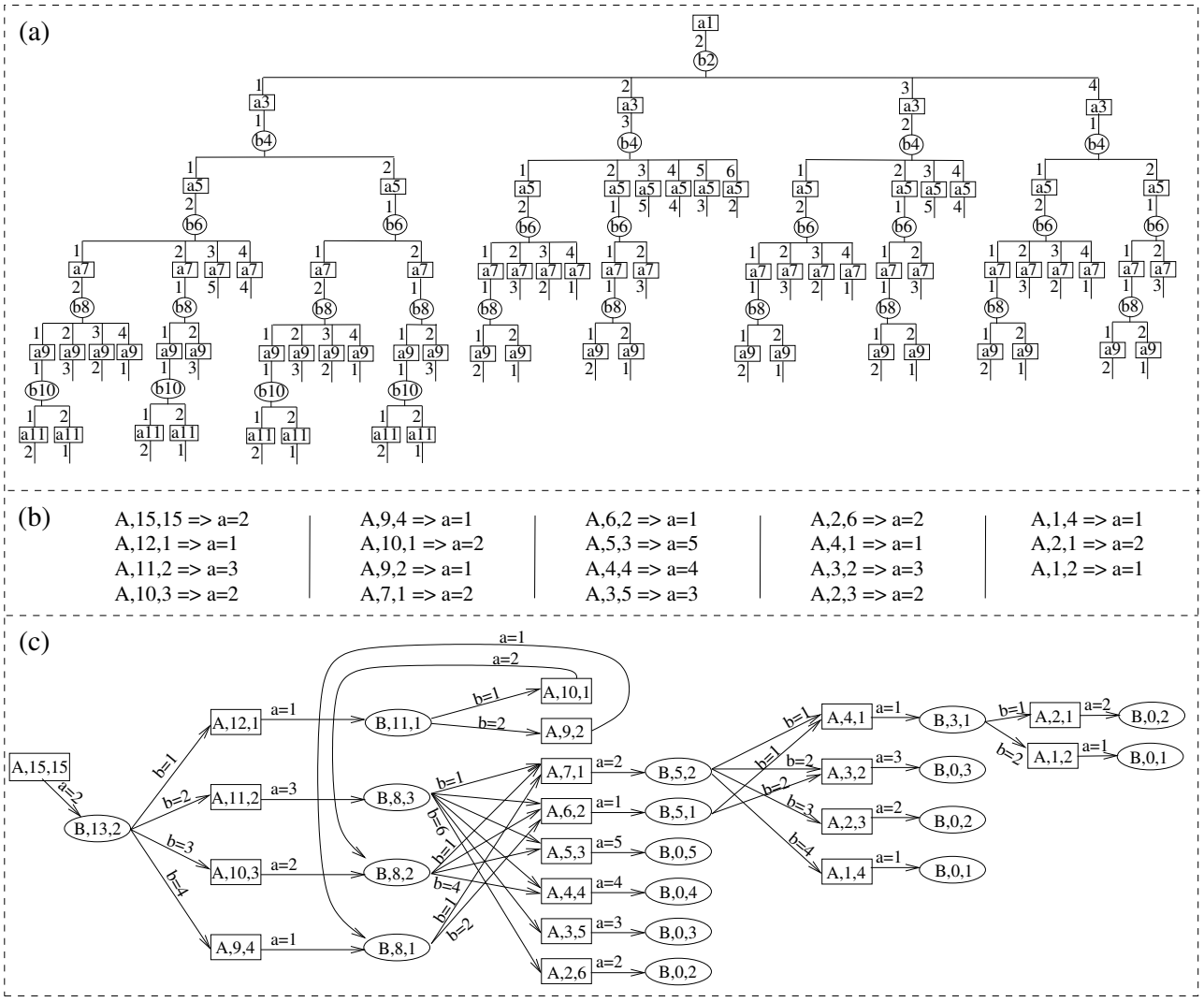


FIGURE 1 – Comparaison entre QCSP et modèle à base d'état sur le jeu *NimFibo* : (a) arbre QCSP de politique ; (b) politique à base d'état ; (c) graphe d'atteignabilité suivant cette politique.

à base d'état peut raisonner sur des horizons non bornés. Elle ne requiert pas que l'évolution du système soit dépliée, ce qui conduit à des modèles plus compacts.

Deuxièmement, on peut observer qu'avec $N = 15$, l'arbre de politique contient 48 feuilles et que la politique à base d'état contient seulement 19 paires (état,décision) alors que les deux induisent les mêmes séquences de décision et d'état. Le rapport en taille croît exponentiellement avec le nombre d'allumettes. La raison principale en est que, dans un contexte d'observabilité complète de l'état et de dynamique markovienne du système, l'arbre de politique mémorise trop d'information. Par exemple, dans le graphe d'atteignabilité de la figure 1(c), les deux séquences de décision $seq_1 : [a = 2, b = 1, a = 1, b = 1, a = 2]$ et $seq_2 : [a = 2, b = 3, a = 2]$ sont équivalentes car elles se terminent sur le même état $(B, 8, 2)$. La seule information utile pour prendre une décision dans cet état est l'état lui-même et non la trajectoire qui a permis de l'atteindre. En d'autres termes, rechercher des stratégies à mémoire complète sous forme d'arbre de politique comme cela est fait dans le cadre QCSP/QCSP+ revient à chercher dans un espace inutilement grand puisqu'il suffit de chercher des stratégies sans mémoire sous forme de politique.

Troisièmement, raisonner explicitement sur l'état permet de mémoriser qu'un état a déjà été exploré et le résultat de cette exploration (succès ou échec) et ainsi d'éviter toute nouvelle exploration. Par exemple, sur la figure 1(c), il n'est pas nécessaire d'explorer deux fois

les trajectoires débutant dans l'état $(B, 8, 2)$ (une fois quand cet état est atteint à partir de l'état $(A, 10, 1)$ et une autre fois quand il l'est à partir de l'état $(A, 10, 3)$). Ces notions de *good/nogood* sur les états ne sont pas gérés par l'approche QCSP qui peut uniquement gérer des *goods/nogoods* sur des ensembles de variables du modèle QCSP. Mémoriser de l'information sur les états déjà explorés utilise les principes de la programmation dynamique en avant [7].

Pour toutes ces raisons, nous pensons qu'il est nécessaire d'introduire un nouveau cadre à base de contraintes faisant explicitement appel à la notion d'état, permettant de modéliser et de résoudre efficacement des problèmes de contrôle de systèmes dynamiques complètement observables et markoviens.

3 Markovian Game CSP (MGCSP)

Dans ce qui suit, on considère que l'état du système est décrit par un ensemble S de variables. Toute affectation $s \in \mathbf{d}(S)$ est appelée un *état* (étant donné un ensemble ordonné X de variables, $\mathbf{d}(X)$ désigne le produit cartésien des domaines des variables de X). On considère de la même façon que les décisions prises à chaque tour par le $\exists$-joueur (décideur ou contrôleur) et le $\forall$-joueur (adversaire ou environnement) sont décrites par deux ensembles C et U de variables. Toute affectation $c \in \mathbf{d}(C)$ (resp. $u \in \mathbf{d}(U)$) est appelée une *décision* contrôlable (resp. incontrôlable).

3.1 Modèle du système

Pour représenter les états initiaux et finaux possibles, nous utilisons une relation d'initialisation $I \subseteq \mathbf{d}(S)$ et une relation de terminaison $E \subseteq \mathbf{d}(S)$. Pour représenter le fait que certaines décisions $c \in \mathbf{d}(C)$ (resp. $u \in \mathbf{d}(U)$) ne peuvent pas être prises dans certains états, du fait de règles de jeu ou de contraintes physiques, nous utilisons une relation de faisabilité $F_c \subseteq \mathbf{d}(S) \times \mathbf{d}(C)$ (resp. $F_u \subseteq \mathbf{d}(S) \times \mathbf{d}(U)$) telle que $F_c(s, c)$ (resp. $F_u(s, u)$) est vrai ssi la décision c (resp. u) est faisable dans l'état s. La dynamique du système est définie par une fonction de transition $T_c : \mathbf{d}(S) \times \mathbf{d}(C) \to \mathbf{d}(S)$ (resp. $T_u : \mathbf{d}(S) \times \mathbf{d}(U) \to \mathbf{d}(S)$) telle que $s' = T_c(s, c)$ (resp. $s' = T_u(s, u)$) signifie que s' est le résultat de l'application de la décision c (resp. u) dans l'état s.

Trois hypothèses sont faites pour garantir l'absence de blocage : premièrement, il existe au moins un état initial possible, c'est-à-dire un état s tel que $I(s)$ est vrai ; deuxièmement, pour tout état s non terminal ($E(s)$ faux), il existe au moins une décision faisable, c'est-à-dire une décision c (resp. u) telle que $F_c(s, c)$ (resp. $F_u(s, u)$) est vrai ; troisièmement, pour tout état s non terminal et pour toute décision c (resp. u) faisable dans s, $T_c(s, c)$ (resp. $T_u(s, u)$) est défini (il peut être non défini pour des décisions infaisables). Ces trois hypothèses ne sont en fait pas très contraignantes : si la première est violée, il est évident que le but ne peut pas être atteint ; si la seconde l'est, il suffit d'ajouter une valeur nulle à chaque variable de C (resp. U) pour garantir que ne rien faire est toujours faisable ; si la troisième l'est, la relation de faisabilité F_c (resp. F_u) peut être restreinte en considérant que les décisions qui n'induisent pas d'état suivant sont infaisables. Les relations I, E, F_c, F_u, T_c et T_u sont exprimées via des ensembles de contraintes. Tous ces éléments sont réunis dans la notion de Markovian Game CSP.

Définition 1 *Un MGCSP (Markovian Game CSP) est un n-uplet* $M = (S, I, E, C, U, F_c, T_c, F_u, T_u)$ *avec :*

- S *un ensemble fini de variables à domaine fini, appelées* variables d'état *;*
- I *un ensemble fini de contraintes sur* S*, appelées* contraintes d'initialisation *;*
- E *un ensemble fini de contraintes sur* S*, appelées* contraintes de terminaison *;*
- C *un ensemble fini de variables à domaine fini, appelées* variables contrôlables *;*
- U *un ensemble fini de variables à domaine fini, appelées* variables incontrôlables *;*
- F_c *et* F_u *des ensembles finis de contraintes respectivement sur* $S \cup C$ *et* $S \cup U$*, appelées* contraintes de faisabilité *;*
- T_c *et* T_u *des ensembles finis de contraintes respectivement sur* $S \cup C \cup S'$ *et* $S \cup U \cup S'$*, appelées* contraintes de transition *;*
- $\exists s \in \mathbf{d}(S),\ I(s)$ *;*
- $\forall s \in \mathbf{d}(S),\ \neg E(s) \to ((\exists c \in \mathbf{d}(C),\ F_c(s, c)) \wedge (\exists u \in \mathbf{d}(U),\ F_u(s, u)))$ *;*
- $\forall s \in \mathbf{d}(S),\ \neg E(s) \to ((\forall c \in \mathbf{d}(C),\ F_c(s, c) \to (\exists! s' \in \mathbf{d}(S),\ T_c(s, c, s'))) \wedge (\forall u \in \mathbf{d}(U),\ F_u(s, u) \to (\exists! s' \in \mathbf{d}(S),\ T_u(s, u, s'))))$

S' est une copie des variables de S représentant la valeur des variables d'état dans l'état suivant ; s' représente l'état suivant.

Pour illustrer cette définition, reconsidérons le jeu *NimFibo*. Dans cet exemple, l'ensemble S des variables d'état est constitué des variables j, r et p précédemment définies, représentant respectivement le joueur courant (valeur dans $\{A, B\}$), le nombre d'allumettes restantes (valeur dans $[0..N]$) et le nombre d'allumettes prises par le joueur précédent (valeur dans $[1..N]$). L'ensemble C (resp. U) des variables contrôlables (resp. incontrôlables) contient une seule variable a (resp. b) représentant le nombre d'allumettes prises par le joueur A (resp. B). Les différents ensembles de contraintes sont indiqués ci-dessous. I exprime que A

joue en premier et qu'il y a au départ N allumettes (la variable p est arbitrairement initialisée avec la valeur N). E exprime que le jeu se termine quand il ne reste plus d'allumettes. F_c et F_u expriment qu'à chaque étape, un joueur ne peut pas prendre plus deux fois le nombre d'allumettes prises par le joueur précédent. T_c et T_u définissent la fonction de transition du système : on change de joueur et le nombre d'allumettes restantes est diminué du nombre d'allumettes prises.

$$I : (j = A) \wedge (r = N) \wedge (p = N)$$
$$E : (r = 0)$$
$$F_c : (a \leq r) \wedge (a \leq 2 \cdot p)$$
$$F_u : (b \leq r) \wedge (b \leq 2 \cdot p)$$
$$T_c : (j' = B) \wedge (r' = r - a) \wedge (p' = a)$$
$$T_u : (j' = A) \wedge (r' = r - b) \wedge (p' = b)$$

3.2 Problèmes d'atteignabilité

Un MGCSP décrit la dynamique du système considéré et induit un ensemble de *trajectoires* possibles.

Définition 2 *Soit $M = (S, I, E, C, U, F_c, T_c, F_u, T_u)$ un MGCSP. L'ensemble des trajectoires induites par M est l'ensemble des séquences (possiblement infinies) d'état et de décision $seq : s_1 \xrightarrow{c_1} s_2 \xrightarrow{u_2} s_3 \xrightarrow{c_3} s_4 \xrightarrow{u_4} s_5 \cdots$ telles que :*

- *$I(s_1)$ est vrai et, pour tout état s_i qui n'est pas le dernier de la séquence, $E(s_i)$ est faux ;*
- *pour toute transition $s_i \xrightarrow{c_i} s_{i+1}$ de seq, $F_c(s_i, c_i)$ et $T_c(s_i, c_i, s_{i+1})$ sont vrais ;*
- *pour toute transition $s_i \xrightarrow{u_i} s_{i+1}$ de seq, $F_u(s_i, u_i)$ et $T_u(s_i, u_i, s_{i+1})$ sont vrais.*

Pour contrôler le système et restreindre ces évolutions possibles, nous utilisons des *politiques* π qui sont des fonctions de l'ensemble $\mathbf{d}(S)$ des états vers l'ensemble $\mathbf{d}(C)$ des décisions : $\pi(s) = c$ spécifie de prendre la décision c dans l'état s. Une politique peut être partielle (non définie pour certains états). Des politiques partielles sont utiles pour définir le contrôle uniquement sur l'ensemble des états atteignables. Nous sommes aussi intéressés par des politiques applicables qui spécifient uniquement des décisions faisables. Ces éléments sont formalisés ci-dessous.

Définition 3 *Une politique pour un MGCSP $M = (S, I, E, C, U, F_c, T_c, F_u, T_u)$ est une fonction partielle $\pi : \mathbf{d}(S) \to \mathbf{d}(C)$. Pour toute politique π, $\mathbf{d}(\pi)$ désigne le domaine de π. Une politique π est applicable ssi pour tout $s \in \mathbf{d}(\pi)$, $F_c(s, \pi(s))$ est vrai.*

L'ensemble des trajectoires induites par une politique π est l'ensemble des trajectoires $seq : s_1 \xrightarrow{c_1} s_2 \xrightarrow{u_2} s_3 \xrightarrow{c_3} s_4 \xrightarrow{u_4} s_5 \cdots$ induites par M et telles que $c_i = \pi(s_i)$ pour toute transition $s_i \xrightarrow{c_i} s_{i+1}$ de seq. Une trajectoire induite par une politique π est complète dans les trois cas suivants :

- *elle est infinie ;*
- *elle est finie et, dans le dernier état s_j, c'est au tour du $\forall$-joueur et $E(s_j)$ est vrai (état terminal) ;*
- *elle est finie et, dans le dernier état s_j, c'est au tour du $\exists$-joueur et $E(s_j)$ est vrai ou $s_j \notin \mathbf{d}(\pi)$ (état terminal ou politique non définie).*

Différentes exigences peuvent être imposées sur les trajectoires. Nous nous focalisons ici sur des exigences d'*atteignabilité* qui imposent que les trajectoires se terminent dans un état satisfaisant une certaine condition.

Définition 4 *Un problème d'atteignabilité est une paire (M, G) avec M un MGCSP sur un ensemble S de variables d'état et G un ensemble fini de contraintes sur S, appelées* contraintes de but. *Une solution de ce problème est une politique applicable π pour M telle que toutes les trajectoires complètes induites par π soient finies et se terminent dans un état s_i tel que $G(s_i)$ est vrai.*

Le problème *NimFibo* est un problème d'atteignabilité (M, G) avec M le MGCSP défini en Section 3.1 et $G : (p = B) \wedge (r = 0)$ (exigence d'atteindre un état où B doit jouer et il ne reste plus d'allumettes). Une politique solution est donnée dans la figure 1(b).

3.3 Relation avec le problème QCSP/QCSP+

Étant donné un problème d'atteignabilité (M, G) et un entier N, considérons le QCSP+ suivant, noté $Q_N(M, G)$, qui pourrait être mis sous forme normale prénexe :

$$\begin{aligned}
Q_N(M,G) : {} & \forall S_1[I(S_1)] \\
& G(S_1) \vee (\neg E(S_1) \wedge \\
& \exists C_1, S_2[F_c(S_1, C_1) \wedge T_c(S_1, C_1, S_2)] \\
& (E(S_2) \wedge G(S_2)) \vee (\neg E(S_2) \wedge \\
& \forall U_2, S_3[F_u(S_2, U_2) \wedge T_u(S_2, U_2, S_3)] \\
& G(S_3) \vee (\neg E(S_3) \wedge \\
& \exists C_3, S_4[F_c(S_3, C_3) \wedge T_c(S_3, C_3, S_4)] \\
& ... \\
& G(S_{N\text{-}1}) \vee (\neg E(S_{N\text{-}1}) \wedge \\
& \exists C_{N\text{-}1}, S_N[F_c(S_{N\text{-}1}, C_{N\text{-}1}) \wedge T_c(S_{N\text{-}1}, C_{N\text{-}1}, S_N)] \\
& (E(S_N) \wedge G(S_N)))...)))
\end{aligned} \tag{1}$$

$Q_N(M, G)$ peut être lu comme : "Est-il vrai que, pour tout état initial possible s_1, soit s_1 est un état

but, soit il n'est pas terminal et il existe une décision faisable c_1 qui induit l'état suivant s_2 tel que, soit s_2 est un état but terminal, soit il n'est pas terminal et pour toute décision faisable u_2 qui induit l'état suivant s_3, soit s_3 est un état but, soit il n'est pas terminal et il existe une décision faisable $c_3 \ldots$ telle que, soit s_{N-1} est un état but, soit il n'est pas terminal et il existe une décision faisable c_{N-1} qui induit l'état suivant s_N qui est un état but terminal ?".

Proposition 1 *Étant donné un problème d'atteignabilité (M, G) et un entier N, il existe une stratégie gagnante pour le QCSP $Q_N(M, G)$ ssi il existe une politique solution π pour (M, G) telle que toutes les trajectoires complètes induites par π soient de longueur inférieure ou égale à N.*

La proposition 1 implique qu'il est possible de résoudre le problème d'atteignabilité (M, G) en résolvant le QCSP $Q_N(M, G)$. Cependant, l'approche n'est pas complète à moins de choisir une valeur de N suffisamment grande : il est possible que le QCSP n'ait pas de solution alors que le problème d'atteignabilité en a. L'approche n'est complète que si on choisit une valeur de N suffisamment grande, par exemple égale au nombre d'états possibles ($N = |\mathbf{d}(S)|$). Mais, comme $|\mathbf{d}(S)|$ peut être très grand et comme le nombre de variables et de contraintes de $Q_N(M, G)$ est linéaire en N, cette approche peut ne pas être concrètement applicable. Cette relation entre problème d'atteignabilité (en contexte non déterministe) et QCSP/QBF est le pendant de la relation existante entre problème d'atteignabilité en contexte déterministe et CSP/SAT [6]. Dans une autre direction, la proposition 1 peut être vue comme le pendant de la propriété des processus de décision markoviens (MDPs [12]) indiquant que tout MDP a une politique optimale stationnaire (ne dépendant que de l'état et non de l'étape).

De plus, en termes d'espace nécessaire pour mémoriser une stratégie gagnante, la taille des politiques peut être exponentiellement plus petite que celle des arbres de politique, ce qui peut être utile quand on veut embarquer un contrôleur à bord d'un système autonome à mémoire limitée. Plus précisément, si R_π désigne l'ensemble des états atteignables en suivant π, la politique π peut être mémorisée comme une table contenant $|R_\pi|$ paires (s, c). Si W est une stratégie gagnante équivalente pour $Q_N(M, G)$ (induisant les mêmes trajectoires que π) représentée sous la forme d'un arbre de politique, la stratégie W peut contenir jusqu'à $|\mathbf{d}(U)|^{N/2}$ feuilles, ce qui est exponentiel en N et toujours supérieur ou égal à $|R_\pi|$.

4 Algorithme

L'algorithme proposé pour résoudre des problèmes d'atteignabilité sur des MGCSP est inspiré de techniques de planification non déterministe [2]. Une différence est l'utilisation de la programmation par contraintes pour raisonner sur les différentes relations.

4.1 Vue globale

L'algorithme est composé de trois fonctions :
- **reachMGCSP** responsable de l'exploration des différents états initiaux possibles ;
- **exploreC** responsable de l'exploration des différentes différentes décisions contrôlables faisables dans un état ;
- **exploreU** qui fait la même chose pour les décisions incontrôlables.

La recherche explore un arbre Et/Ou dans lequel les nœuds Ou correspondent aux décisions contrôlables et les nœuds Et aux décisions incontrôlables. L'arbre est exploré en profondeur d'abord et seuls les états atteignables à partir des états initiaux sont considérés.

Au cours de la recherche, l'algorithme maintient une politique courante π. Il associe à chaque état s une marque $Mark(s) \in \{Solved, Bad, Processing, None\}$. Une marque *Solved* signifie que l'état s a déjà été visité et que la politique courante permet d'atteindre à coup sûr le but en partant de s. Une marque *Bad* signifie qu'il n'existe aucune politique permettant d'atteindre à coup sûr le but en partant de s. Une marque *Processing* est associé aux états en cours d'exploration. Une marque *None*, qui reste implicite, est associé aux autres états. Dans l'implémentation existante, les marques sont mémorisées dans une table de *hashage* initialement vide (tous les états avec la marque *None*). On suppose l'existence d'une variable d'état booléenne indiquant si c'est au tour du $\exists$-joueur ou du $\forall$-joueur.

Une spécificité de l'algorithme concerne la gestion des *boucles*. Une boucle est une situation dans laquelle un état marqué *Processing* est de nouveau rencontré. Quand une boucle est rencontrée, cela signifie que l'adversaire (nature ou autre joueur) peut générer une trajectoire qui boucle indéfiniment sans que le but soit atteint. Par exemple, supposons que la figure 2 représente l'ensemble des trajectoires possibles d'un système et que le but soit d'atteindre l'état s_f. Les trajectoires $seq_1 : s_a \overset{c:0}{\rightarrow} s_b \overset{u:0}{\rightarrow} s_c \overset{c:0}{\rightarrow} s_d \overset{u:1}{\rightarrow} s_e \overset{c:0}{\rightarrow} s_b$ et $seq_2 : s_a \overset{c:0}{\rightarrow} s_b \overset{u:0}{\rightarrow} s_c \overset{c:0}{\rightarrow} s_d \overset{u:1}{\rightarrow} s_e \overset{c:1}{\rightarrow} s_d$ bouclent respectivement sur s_b et s_d. En conséquence, la trajectoire $seq_3 : s_a \overset{c:0}{\rightarrow} s_b \overset{u:0}{\rightarrow} s_c \overset{c:0}{\rightarrow} s_d \overset{u:1}{\rightarrow} s_e$ ne peut pas être étendue en une solution. L'ensemble $J = \{s_b, s_d\}$ est appelé la *justification de bouclage* de seq_3. Il correspond à l'ensemble des états précédents sur lesquels des

boucles ont été détectées en essayant d'étendre seq_3. La marque de s_e, le dernier état de seq_3, ne peut cependant pas être positionnée à *Bad* parce que les boucles détectées dépendent de décisions prises avant s_e. Pour $seq_4 : s_a \overset{c:0}{\rightarrow} s_b \overset{u:0}{\rightarrow} s_c \overset{c:0}{\rightarrow} s_d$ et $seq_5 : s_a \overset{c:0}{\rightarrow} s_b \overset{u:0}{\rightarrow} s_c$, la justification de bouclage est $\{s_b\}$. Pour $seq_6 : s_a \overset{c:0}{\rightarrow} s_b$, elle est vide. La marque de s_b peut donc être positionnée à *Bad*. Plus généralement, un état dont l'exploration échoue peut être marqué *Bad* si la justification courante de bouclage est vide.

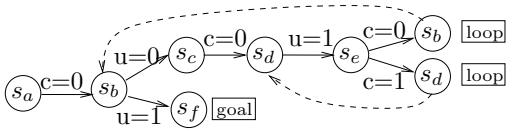


FIGURE 2 – Boucles dans la dynamique d'un système.

4.2 Pseudo-code

La fonction principale **reachMGCSP** prend en entrée un MGCSP M et un ensemble G de contraintes de but. Il retourne $(true, \pi)$ si (M, G) admet une politique solution π et $(false, \varnothing)$ sinon. Pour cela, la fonction **reachMGCSP** démarre avec une politique vide et analyse chaque état initial possible (la fonction *getSols* appelée en ligne 6 retourne l'ensemble des solutions d'un CSP et donc ici l'ensemble des états initiaux possibles). Chaque état initial s qui n'est pas un état but et dont la marque est différente de *Solved* est ensuite analysé. Si s est terminal ou est marqué *Bad*, le problème n'a pas de solution et $(false, \varnothing)$ est retourné (ligne 8). Sinon s est exploré par appel à la fonction **exploreC** (ligne 10).

```
1  Entrée : un MGCSP M et un ensemble G de
   contraintes de but
2  Sortie : une paire (b, π) avec b un booléen et π une
   politique
3  reachMGCSP(M, G)
4  begin
5      π ← ∅
6      foreach s ∈ getSols(I(S)) do
7          if ¬G(s) ∧ (Mark(s) ≠ Solved) then
8              if E(s) ∨ (Mark(s) = Bad) then return
               (false, ∅)
9              else
10                 (covered, π, .) ← exploreC(s, π)
11                 if ¬covered then return (false, ∅)
12     return (true, π)
```

La fonction **exploreC**(s, π) explore les décisions qui peuvent être prises dans l'état s. Elle retourne un triplet (b, π', J). b indique si la politique π peut être étendue de façon à ce que le but soit atteint à coup sûr depuis s. Si $b = true$, π' est la politique étendue. Si $b = false$, J est un ensemble d'états justifiant l'absence de solution à partir de s. J correspond à la justification de bouclage décrite précédemment. La première partie de **exploreC** (lignes 5 à 12) détermine toutes les décisions c qui sont faisables dans l'état s et tous les états suivants associés s' en raisonnant sur le CSP $F_c(S, C) \wedge T_c(S, C, S') \wedge (S = s)$. Si un état suivant s' obtenu en appliquant une décision c est un état but terminal ou est marqué *Solved*, alors il suffit de fixer $\pi(s) = c$ pour couvrir l'état s. La seconde partie de **exploreC** (lignes 13 à 34) parcourt l'ensemble des états suivants s' restant à explorer. Si s' est marqué *Solved*, alors l'état s est couvert (lignes 19 à 21). Sinon, si s' est marqué *Processing*, la justification de bouclage est étendue (lignes 22 et 23). Sinon, si s' est marqué *None*, il est exploré par appel à la fonction **exploreU** (ligne 26). Si cet appel retourne une politique solution étendue, s est marqué *Solved* et la politique étendue est retournée (lignes 27 à 29). Sinon, la justification de bouclage est étendue (ligne 30). Si tous les états suivants s' ont été explorés sans trouver de solution, alors s est retiré de la justification de bouclage, il est marqué *Bad* si la justification de bouclage est vide et un échec est retourné (lignes 31 à 34).

Le comportement de la fonction **exploreU** est similaire. Les seules différences sont les suivantes. Dans la première partie (lignes 5 à 15), un échec est retourné s'il existe un état suivant s' qui est un état non-but terminal ou qui est marqué *Bad*. Un échec est aussi retourné si une boucle est détectée. Dans la seconde partie (lignes 16 à 31), dans laquelle les états suivants restants sont explorés, si un état suivant s' est marqué *Bad*, un échec est retourné (lignes 20 à 22). Sinon, la fonction **exploreC** est appelée pour développer les différentes décisions faisables en s' (ligne 24).

Proposition 2 *L'algorithme* **reachMGCSP** *est correct et complet : il retourne* $(true, \pi)$ *avec* π *une politique solution si le problème d'atteignabilité* (M, G) *admet une solution et* $(false, \varnothing)$ *sinon.*

4.3 Améliorations algorithmiques

L'algorithme de base peut être amélioré sur différents points. Tout d'abord, dans la fonction **exploreC**, le CSP $F_c(S, C) \wedge T_c(S, C, S') \wedge G(S') \wedge E(S') \wedge (S = s)$ peut être considéré pour déterminer rapidement s'il existe une décision contrôlable permettant d'atteindre directement un état but terminal plutôt que d'énumérer toutes les solutions de $F_c(S, C) \wedge T_c(S, C, S') \wedge (S = s)$ et de vérifier en-

```
1  Entrée : un état s et une politique courante π
2  Sortie : un triplet (b, π', J) avec b un booléen, π' une
     politique étendue et J une justification de bouclage
3  exploreC(s, π)
4  begin
5      toExplore ← ∅
6      foreach
       sol ∈ getSols(F_c(S,C) ∧ T_c(S,C,S') ∧ (S = s)) do
7          (s, c, s') ← (sol^↓S, sol^↓C, sol^↓S')
8          if (G(s') ∧ E(s')) ∨ (Mark(s') = Solved) then
9              setMark(s, Solved) ;
10             return (true, π ∪ {(s, c)}, ∅)
11         else if ¬E(s') then
12             toExplore ← toExplore ∪ {(c, s')}
13     π' ← π
14     setMark(s, Processing)
15     J ← ∅
16     while toExplore ≠ ∅ do
17         Choose (c, s') ∈ toExplore ;
18         toExplore ← toExplore \ {(c, s')}
19         if Mark(s') = Solved then
20             setMark(s, Solved) ;
21             return (true, π' ∪ {(s, c)}, ∅)
22         else if Mark(s') = Processing then
23             J ← J ∪ {s'}
24         else if Mark(s') = None then
25             π' ← π' ∪ {(s, c)}
26             (covered, π', J') ← exploreU(s', π')
27             if covered then
28                 setMark(s, Solved) ;
29                 return (true, π', ∅)
30             else J ← J ∪ J' ; π' ← π' \ {(s, c)}
31     J ← J \ {s}
32     if J = ∅ then setMark(s, Bad)
33     else setMark(s, None)
34     return (false, π', J)
```

```
1  Entrée : un état s et une politique courante π
2  Sortie : un triplet (b, π', J) avec b un booléen, π' une
     politique étendue et J une justification de bouclage
3  exploreU(s, π)
4  begin
5      toExplore ← ∅
6      foreach
       sol ∈ getSols(F_u(S,U) ∧ T_u(S,U,S') ∧ (S = s)) do
7          s' ← sol^↓S'
8          if ¬G(s') then
9              if E(s') ∨ (Mark(s') = Bad) ∨ (s = s')
               then
10                 setMark(s, Bad) ;
11                 return (false, π, ∅)
12             else if Mark(s') = Processing then
13                 return (false, π, {s'})
14             else if Mark(s') = None then
15                 toExplore ← toExplore ∪ {s'}
16     π' ← π
17     setMark(s, Processing)
18     while toExplore ≠ ∅ do
19         Choose s' ∈ toExplore ;
           toExplore ← toExplore \ {s'}
20         if Mark(s') = Bad then
21             setMark(s, Bad) ;
22             return (false, π', ∅)
23         else if Mark(s') = None then
24             (covered, π', J) ← exploreC(s', π')
25             if ¬covered then
26                 J ← J \ {s}
27                 if J = ∅ then setMark(s, Bad)
28                 else setMark(s, None)
29                 return (false, π', J)
30     setMark(s, Solved) ;
31     return (true, π', ∅)
```

suite si l'une d'elles satisfait $G(S') \wedge E(S')$. De façon similaire, il est possible de considérer le CSP $F_u(S,U) \wedge T_u(S,U,S') \wedge \neg G(S') \wedge E(S') \wedge (S = s)$ dans la fonction **exploreU** pour déterminer rapidement s'il existe une décision incontrôlable conduisant directement à un état non-but terminal.

En termes d'espace mémoire, l'algorithme mémorise les marques uniquement sur les états atteignables, mais ces états peuvent être très nombreux et la mémorisation coûteuse. Pour contourner cette difficulté, certaines marques peuvent être oubliées au cours de la recherche. L'algorithme reste valide, mais peut réexplorer certaines parties de l'espace de recherche. Cette option n'a pas été utilisée dans les expérimentations.

Finalement, nous considérons dans le cadre MGCSP que le ∃-joueur commence. Pour traiter des situations dans lesquelles le ∀-joueur commence, il suffit de remplacer l'appel à **exploreC** dans la fonction **reachMGCSP** par un appel à **exploreU**.

5 Expérimentations

Nous avons mené nos expérimentations sur un processeur Intel i5-520, 1.2GHz, 4GB RAM. Le temps de calcul maximum a été fixé à une heure. L'algorithme **reachMGCSP** est implémenté dans *Dyncode*, un outil développé au dessus de la librairie *Gecode*[1] de programmation par contraintes. Toute contrainte disponible dans *Gecode* peut être utilisée dans *Dyncode*. *Dyncode* a été initialement conçu pour traiter des problèmes de contrôle en environnement non déterministe et partiellement observable [11]. L'algorithme **reachMGCSP**, qui suppose un état complètement

1. `http://www.gecode.org/`

observable, est cependant plus efficace que les algorithmes décrits dans [11]. Dans les expérimentations, nous avons utilisé *min-domain* pour l'heuristique de choix de variable et un ordre lexicographique pour le choix de valeur.

Nous avons d'abord comparé *Dyncode* avec *Qecode*[2], un solveur de QCSP+ construit sur *Gecode*. Nous avons mené des expérimentations sur trois jeux déjà modélisés dans la distribution *Qecode* : *NimFibo*, *Connect4* et *MatrixGame*. Les figures 3(a) à 3(c) montrent que, sur ces problèmes, *Dyncode* est supérieur à *Qecode* de plusieurs ordres de grandeur.

NimFibo Pour *NimFibo* (figure 3(a)), *Qecode* peut résoudre les instances jusqu'à 40 allumettes alors que *Dyncode* peut résoudre des instances impliquant plusieurs dizaines de milliers d'allumettes. La complexité temporelle observée avec *Dyncode* est même linéaire en fonction du nombre d'allumettes alors qu'on observe une évolution exponentielle avec *Qecode*. En d'autres termes, utiliser explicitement la notion d'état et mémoriser au cours de la recherche les états *good/nogood* casse la complexité.

Connect4 *Connect4* est un jeu à deux joueurs sur un tableau à 6 lignes et 7 colonnes. À chaque étape, un joueur place un jeton dans une colonne du tableau. Par gravité, le jeton tombe au bas de la colonne. Un joueur gagne s'il réussit à aligner 4 de ses jetons horizontalement, verticalement ou en diagonale. Le résultat du jeu est nul si le tableau est plein et sans aucun alignement. Comme dans la distribution *Qecode*, nous considérons une variante de ce jeu appelée *Connect4_Bounded* dont le but est de jouer sans perdre sur un nombre fixé d'étapes. La figure 3(b) montre que *Dyncode* résout plus d'instances que *Qecode*. Cette fois, on observe une évolution exponentielle avec les deux solveurs, mais l'évolution est plus lente pour *Dyncode*. La première explication est encore l'utilisation explicite de la notion d'état puisque, dans *Connect4*, plusieurs séquences de jeu peuvent conduire à la même configuration. La seconde est que *Qecode* crée initialement de nombreuses variables et contraintes pour déplier le problème sur l'horizon du jeu. Il peut ensuite réaliser ce qui est appelé une propagation en cascade sur le problème complet. A l'opposé, *Dyncode* propage les contraintes uniquement sur l'état courant et sur le suivant. Cela peut produire moins d'effacements, mais est réalisé beaucoup plus rapidement.

MatrixGame Dans le jeu *MatrixGame*, une matrice 0/1 de taille 2^d est considérée. À chaque tour, le $\exists$-joueur coupe la matrice horizontalement et garde le haut ou le bas. Le $\forall$-joueur coupe alors la matrice verticalement et garde la droite ou la gauche. Après d coups, la matrice est réduite à une case. Le $\exists$-joueur gagne si cette case contient la valeur 1. La figure 3(c) montre que *Dyncode* se comporte encore une fois mieux que *Qecode*, même si le modèle MGCSP est tel qu'un état ne peut pas être visité plus d'une fois. Nous pensons que c'est toujours parce que *Dyncode* ne raisonne que sur des CSP de petite taille.

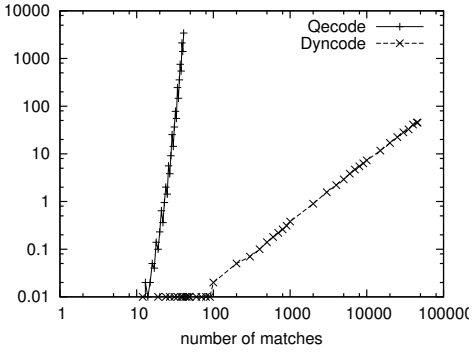
Dyncode a aussi été comparé avec *Queso*, un solveur de QCSP a priori plus efficace que *BlockSolve* or *QCSP-solve* [9]. Nous n'avons pas relancé *Queso* qui n'est plus maintenu et nous avons retenu les résultats indiqués dans [9], obtenus avec un processeur Pentium 4, 3.06GHz, 1GB RAM. Les résultats sont indiqués dans la figure 3(d) pour *Connect4*, mais cette fois avec des tableaux de taille $N \times M$ et l'objectif que le $\exists$-joueur remporte le jeu. Les résultats montrent que *Dyncode* est plus performant que *Queso*. Encore une fois, nous pensons que la notion d'état est ici vraiment utile pour éviter d'explorer plusieurs fois la même partie de l'espace de recherche. Alors que *Queso* utilise des techniques telles que la règle de valeur pure et la propagation de contraintes sur des contraintes disjonctives réifiées, la propagation de contraintes limitée réalisée par *Dyncode* limite le temps de calcul.

Dyncode pourrait être comparé avec d'autres outils : (a) des outils de planification non déterministe comme, par exemple, MBP [2], (b) des outils de synthèse de contrôleur venant de la communauté *model checking* [10] ou des outils de résolution de MDPs [12], en considérant les MGCSP comme des MDP avec des probabilités 0/1. Les algorithmes de résolution de MDP qui explorent tout l'espace d'état, tels que les algorithmes d'itération de la valeur ou de la politique, risquent d'être rapidement inefficaces. Des algorithmes de recherche dans un arbre Et/Ou peuvent être plus compétitifs, mais leur gestion des probabilités et des *backups* peut être pénalisante. Ces comparaisons expérimentales restent à faire. Un point important est le fait que ces outils n'offrent pas la flexibilité des modèles à base de contraintes.

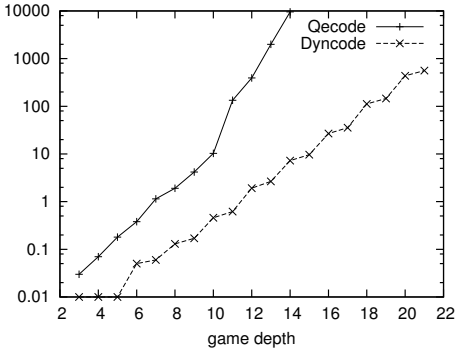
6 Conclusion

Ce papier a montré que, pour l'instant, utiliser le cadre QCSP/QCSP+ n'est pas la meilleure approche à base de contraintes pour modéliser et résoudre des problèmes de contrôle de systèmes dynamiques satisfaisant les hypothèses de dynamique markovienne et d'observabilité complète. L'utilisation de solveurs QCSP/QCSP+ est certainement plus appropriée pour résoudre des problèmes dans lesquels ces hypothèses ne sont pas vérifiées ou dans lesquels le nombre d'al-

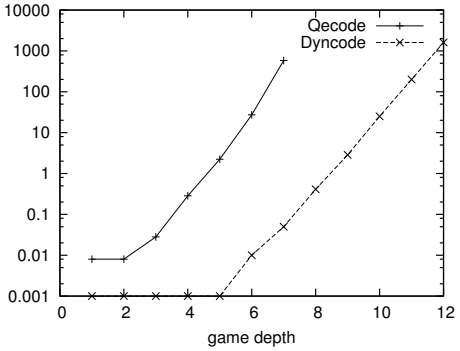
2. `http://www.univ-orleans.fr/lifo/software/qecode`



(a) *NimFibo*



(b) *Connect4_Bounded*



(c) *MatrixGame*

Taille du tableau		Temps de calcul (sec.)	
nCols	nLigns	*Queso* 3.06GHz,1GB RAM	*Dyncode* 1.2GHz,4GB RAM
4	4	1.05	0.44
4	5	9.13	1.47
5	4	63.57	6.44
5	5	1749.94	64.9
5	6	16012.50	1621.8

(d) *Connect4_Full*

FIGURE 3 – Comparaison des temps de calcul obtenus avec *Dyncode*, *Qecode* (solveur QCSP+) et *Queso* (solveur QCSP) sur les jeux (a) *NimFibo*, (b) *Connect4_Bounded*, (c) *MatrixGame* et (d) *Connect4_Full*. Le temps de calcul en secondes est représenté en ordonnée sur une échelle logarithmique.

ternances de quantificateur reste limité. Pour le futur, nous envisageons d'étendre le cadre MGCSP pour modéliser des problèmes de contrôle dans lesquels le nombre de transitions incontrôlables entre deux étapes contrôlables n'est pas fixé. Des exigences sur les trajectoires plus générales que l'atteignabilité pourraient aussi être considérées.

Références

[1] Marco Benedetti, Arnaud Lallouet, and Jérémie Vautard. QCSP Made Practical by Virtue of Restricted Quantification. In *Proc. of IJCAI-07*, pages 38–43, 2007.

[2] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In *Proc. of IJCAI-01*, pages 473–478, 2001.

[3] L. Bordeaux and E. Monfroy. Beyond NP : Arc-consistency for Quantified Constraints. In *Proc. of CP-02*, pages 371–386, 2002.

[4] Ian P. Gent, Peter Nightingale, and Andrew Rowley. Encoding Quantified CSPs as Quantified Boolean Formulae. In *Proc. of ECAI-04*, pages 176–180, 2004.

[5] Ian P. Gent, Peter Nightingale, and Kostas Stergiou. QCSP-Solve : A Solver for Quantified Constraint Satisfaction Problems. In *Proc. of IJCAI-05*, pages 138–143, 2005.

[6] H. Kautz and B. Selman. Planning as Satisfiability. In *Proc. of ECAI-92*, pages 359–363, 1992.

[7] R. Larson and J. Casti. *Principles of Dynamic Programming*. M. Dekker Inc., 1978.

[8] Nikos Mamoulis and Kostas Stergiou. Algorithms for Quantified Constraint Satisfaction Problems. In *Proc. of CP-04*, pages 752–756, 2004.

[9] Peter Nightingale. Non-binary Quantified CSP : Algorithms and Modelling. *Constraints*, 14(4) :539–581, 2009.

[10] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of Reactive(1) Designs. In *Proc. of the 7th International Conference on Verification, Model Checking and Abstract Interpretation*, pages 364–380, 2006.

[11] C. Pralet, G. Verfaillie, M. Lemaître, and G. Infantes. Constraint-Based Controller Synthesis in Non-Deterministic and Partially Observable Domains. In *Proc. of ECAI-10*, pages 681–686, 2010.

[12] M. Puterman. *Markov Decision Processes, Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[13] Kostas Stergiou. Repair-based Methods for Quantified CSPs. In *Proc. of CP-05*, pages 652–666, 2005.

[14] Guillaume Verger and Christian Bessiere. Blocksolve : a Bottom-up Approach for Solving Quantified CSPs. In *Proc. of CP-06*, pages 635–649, 2006.

Réseaux temporels simples étendus Application à la gestion de satellites agiles

Cédric Pralet **Gérard Verfaillie**

ONERA – The French Aerospace Lab, F-31055, Toulouse, France
cedric.pralet@onera.fr gerard.verfaillie@onera.fr

Résumé

Les réseaux temporels simples (STN, Simple Temporal Networks) permettent de représenter une conjonction de contraintes de distance minimale et maximale requises sur certains couples de positions temporelles. Ce papier propose une extension des STN incluant des contraintes temporelles plus générales, pour lesquelles la distance à respecter entre deux positions temporelles x et y n'est plus forcément constante mais peut dépendre des instanciations de x et de y. De telles contraintes sont utiles pour traiter des problèmes dans lesquels le temps de transition entre deux activités peut dépendre de l'instant d'activation de la transition. L'extension proposée est appelée le cadre des "Time-dependent STN" (TSTN). Les propriétés de ce cadre sont étudiées et les techniques de résolution classiques sur les STN sont étendues aux TSTN. Les contributions proposées sont intégrées dans un algorithme de recherche locale utilisé pour la gestion de satellites dits agiles.

1 Motivations

La gestion des aspects temporels est un point clé dans le traitement d'applications issues du domaine de l'ordonnancement ou de la planification. Ces domaines mettent en effet généralement en jeu des contraintes sur les dates de début au plus tôt et de fin au plus tard de certaines activités, des contraintes de précédence ou de non recouvrement entre activités, ou encore des contraintes de distance temporelle requise entre activités. Ces contraintes peuvent dans de nombreux cas être exprimées (1) ou bien comme des contraintes temporelles dites *simples*, de la forme $x - y \in [\alpha, \beta]$, avec x, y des variables correspondant à deux positions temporelles et α, β des constantes ; (2) ou bien comme des contraintes temporelles dites *disjonctives*, correspondant à une disjonction de contraintes temporelles simples. Une conjonction de contraintes temporelles simples peut être représentée sous la forme d'un STN (Simple Temporal Network [6]). Les contraintes temporelles disjonctives peuvent quant à elles être représentées sous la forme d'un DTN (Disjunctive Temporal Network [18]).

Un des intérêts du cadre STN est la complexité polynomiale de certains problèmes [6], notamment : (a) la détermination de la cohérence d'un STN, c'est-à-dire l'existence d'une instanciation des variables temporelles satisfaisant toutes les contraintes, (b) la détermination des dates au plus tôt / au plus tard associées à chaque variable temporelle, utile pour maintenir un ordonnancement avec flexibilité temporelle, et (c) la détermination du *minimal network* [13] associé au STN, donnant les distances temporelles minimale et maximale séparant deux variables temporelles quelconques du STN. Un autre intérêt des STN est qu'ils sont souvent (voire toujours) utilisés comme élément de base dans la résolution des DTN ou de réseaux temporels plus complexes.

Dans ce papier, nous mettons à profit les différentes techniques développées dans le cadre STN pour traiter une application issue du domaine spatial. Cette application, dans laquelle les aspects temporels sont fortement présents, correspond à la gestion de satellites d'observation de la Terre tels que ceux du système *Pléiades*. De tels satellites sont situés en orbite basse, à quelques centaines de kilomètres d'altitude. Ils embarquent un instrument d'observation fixe à bord et sont dits *agiles*, ce qui signifie qu'ils ont la capacité de se mouvoir autour de leurs trois axes de rotation (roulis, tangage, lacet). Cette agilité leur permet de pointer à gauche, à droite, en avant ou en arrière du point au sol à la verticale du satellite. La mission de ces satellites consiste à réaliser des prises de vue de polygones à la surface du globe. Ces polygones sont découpés en bandes devant être balayées par l'instrument d'obser-

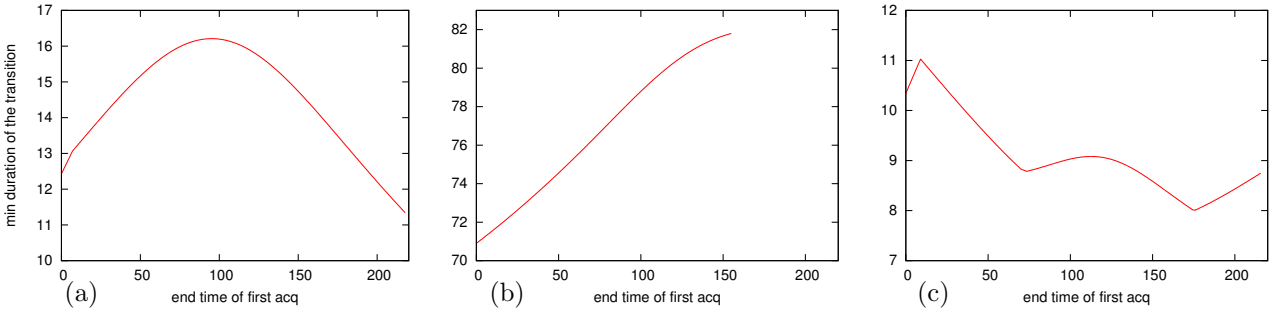


FIGURE 1 – Durées minimales, en secondes, de basculement d'une zone i se terminant au point de latitude-longitude 41°17′48″N-2°5′12″E à une zone j commençant au point de latitude-longitude 42°31′12″N-2°6′15″E, pour différents angles de balayage par rapport à la trace au sol du satellite : (a) balayage de i à 40° et balayage de j à 20° ; (b) balayage de i à 40° et balayage de j à -80° ; (c) balayage de i à 90° et balayage de j à 82°

vation. Le problème d'ordonnancement de prises de vue pour un satellite agile a notamment donné lieu au challenge ROADEF 2003 [16], pour lequel certaines simplifications par rapport au problème réel ont été adoptées, la principale étant que les durées nécessaires au basculement du satellite entre deux prises de vue i et j sont précalculées et considérées comme constantes. Dans ce cas, l'exécutabilité des ordonnancements produits n'est garantie que si les durées constantes choisies sont des majorants des durées réelles.

En pratique, l'hypothèse de temps de transition constants entre deux prises de vue n'est pas vérifiée [12]. Comme illustré à la figure 1, la durée minimale de basculement requise entre la fin d'une prise de vue i et le début d'une prise de vue j dépend de la date à laquelle la prise de vue i se termine. Pour la figure 1(b), l'absence de point sur la courbe après la date $t = 150$ correspond au fait qu'après cette date, la transition de i vers j n'est plus possible. Ces figures montrent la grande variabilité des temps de transition (jusqu'à une dizaine de secondes ici, durée pendant laquelle le satellite parcourt entre 50 et 100km au sol). Elles illustrent également la diversité des schémas d'évolution des temps de transitions minimaux. Ces temps sont calculés par résolution d'un problème d'optimisation de commande continue prenant en compte le mouvement du satellite sur son orbite, le déplacement des points au sol du fait de la rotation de la Terre et des contraintes sur la cinématique du satellite.

Pour toutes ces raisons, il apparaît utile de définir un nouveau cadre permettant de modéliser des transitions entre activités dont la durée dépend de la date d'enclenchement de la transition. Cet aspect se rapproche des travaux effectués autour du *time-dependent scheduling* [5, 7], dans lequel les durées de transition prennent des formes particulières, constantes par morceaux ou linéaires par morceaux, non directement utilisables ici. Nous décrivons tout d'abord le cadre proposé (les Time-dependent STN) ainsi que ses propriétés (sect. 2). Les preuves ne sont pas incluses par manque de place. Des techniques sont ensuite introduites pour calculer les dates de réalisation au plus tôt et au plus tard associées à chaque variable temporelle (sect. 3). Ces techniques sont utilisées au sein d'un algorithme de recherche locale utilisé pour l'ordonnancement d'activités d'un satellite agile (sect. 4).

2 Réseaux temporels simples étendus

Nous rappelons tout d'abord quelques définitions associées aux STN. Dans tout ce qui suit, le domaine d'une variable x est noté $\mathbf{d}(x)$.

2.1 Réseaux temporels simples

Définition 1. *Un réseau temporel simple (STN) est un couple (V, C) avec V un ensemble fini de variables dont le domaine est un intervalle fermé $[l, u] \subset \mathbb{R}$ et C un ensemble fini de contraintes binaires de la forme $x - y \in [\alpha, \beta]$ avec $x, y \in V$, $\alpha \in \mathbb{R} \cup \{-\infty\}$ et $\beta \in \mathbb{R} \cup \{+\infty\}$. Ces contraintes sont appelées des contraintes temporelles simples.*

Une solution d'un STN (V, C) est une instanciation des variables de V qui satisfait toutes les contraintes de C. Un STN est dit cohérent si et seulement si il admet au moins une solution.

Les contraintes unaires $x \in [\alpha, \beta]$, y compris celles définissant les domaines de valeurs possibles des différentes variables, peuvent se reformuler comme des contraintes temporelles simples $x - x_0 \in [\alpha, \beta]$, avec x_0 une variable de domaine $[0, 0]$ jouant le rôle de position temporelle de référence. Par ailleurs, comme $x - y \in [\alpha, \beta]$ équivaut à $(x - y \leq \beta) \wedge (y - x \leq -\alpha)$, il est

possible de se ramener uniquement à des contraintes de la forme $y - x \leq c$ avec c une constante.

Un élément important associé à un STN est son *graphe de distance*. Ce graphe contient un nœud par variable du STN et, pour chaque contrainte $y - x \leq c$ du STN, un arc orienté de x vers y et pondéré par c. Sur la base de ce graphe de distance, il est possible de montrer les résultats suivants [6] :

1. un STN est cohérent si et seulement si il n'existe pas de cycle de longueur négative dans son graphe de distance ;
2. si d_{0i} (resp. d_{i0}) désigne la longueur du plus court chemin dans le graphe de distance du nœud référence étiqueté par x_0 à un nœud étiqueté par une variable temporelle x_i (resp. de x_i à x_0), alors l'ensemble des instanciations cohérentes de x_i correspond à l'intervalle $[-d_{i0}, d_{0i}]$; les plus courts chemins peuvent être calculés pour tout i via l'algorithme de Bellman-Ford ou via des algorithmes s'apparentant à de la propagation de contraintes par arc-cohérence [3, 4, 8, 11] ;
3. si l'on note d_{ij} la longueur du plus court chemin de x_i à x_j dans le graphe de distance et d_{ji} la longueur du plus court chemin de x_j à x_i, alors l'intervalle $[-d_{ji}, d_{ij}]$ représente l'ensemble des distances temporelles possibles entre x_i et x_j ; de tels plus courts chemins peuvent être déterminés pour tous i, j en utilisant l'algorithme de Floyd-Warshall ou des algorithmes basés sur de la propagation par chemin-cohérence [6, 19, 15, 14].

Exemple Nous considérons un problème simplifié d'ordonnancement d'activités d'un satellite agile. Ce problème fait intervenir 3 acquisitions acq_1, acq_2, acq_3 devant être réalisées dans l'ordre $acq_3 \rightarrow acq_1 \rightarrow acq_2$. Pour tout $i \in [1..3]$, on note $Tmin_i$ et $Tmax_i$ les dates de début au plus tôt et de fin au plus tard de acq_i. La durée de acq_i est notée Da_i. Les durées de transition minimales entre la fin de acq_3 et le début de acq_1, et entre la fin de acq_1 et le début de acq_2, sont notées respectivement $Dt_{3,1}$ et $Dt_{1,2}$. Ces durées sont supposées constantes dans cette version simplifiée. Nous considérons également deux fenêtres temporelles $w_1 = [Ts_1, Te_1]$ et $w_2 = [Ts_2, Te_2]$ au cours desquelles un vidage de données vers des stations sol est possible. Le satellite doit vider acq_2 puis acq_3 dans la fenêtre w_1, avant de vider acq_1 dans w_2. La durée de vidage d'une acquisition acq_i est notée Dv_i.

Ce problème peut être modélisé sous la forme d'un STN contenant, pour toute acquisition acq_i ($i \in [1..3]$), les variables temporelles suivantes :

- une date de début de réalisation sa_i et une date de fin de réalisation ea_i, avec comme domaines de valeurs $\mathbf{d}(sa_i) = \mathbf{d}(ea_i) = [Tmin_i, Tmax_i]$;
- une date de début de vidage sv_i et une date de fin de vidage ev_i, avec comme domaines de valeurs $[Ts_1, Te_1]$ pour $i = 2, 3$ et $[Ts_2, Te_2]$ pour $i = 1$.

Les contraintes temporelles simples 1 à 7 sont imposées sur ces variables. Les contraintes 1 et 2 définissent la durée des acquisitions et des vidages. Les contraintes 3 et 4 imposent des temps de transition minimaux entre acquisitions. Les contraintes 5 et 6 assurent le non chevauchement des activités de vidages. La contrainte 7 exprime qu'une acquisition ne peut être vidée qu'après la fin de sa réalisation. La figure 2 donne le graphe de distance associé à cet exemple.

$$\forall i \in [1..3],\ ea_i - sa_i = Da_i \quad (1)$$
$$\forall i \in [1..3],\ ev_i - sv_i = Dv_i \quad (2)$$
$$sa_1 - ea_3 \geq Dt_{3,1} \quad (3)$$
$$sa_2 - ea_1 \geq Dt_{1,2} \quad (4)$$
$$sv_3 - ev_2 \geq 0 \quad (5)$$
$$sv_1 - ev_3 \geq 0 \quad (6)$$
$$\forall i \in [1..3],\ sv_i - ea_i \geq 0 \quad (7)$$

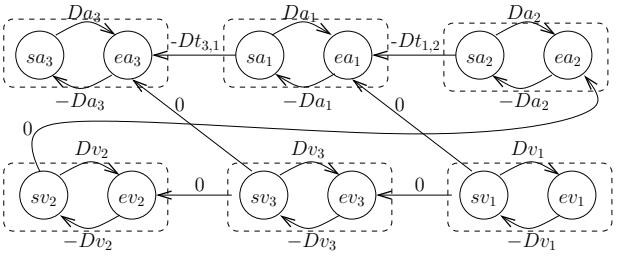


FIGURE 2 – Graphe de distance (la position temporelle de référence x_0 n'est pas représentée)

2.2 Contraintes temporelles t-simples

Nous introduisons une nouvelle classe de contraintes temporelles appelées des *contraintes temporelles t-simples*. Ces dernières sont plus générales que les contraintes temporelles simples et permettent de manipuler des temps de transition entre activités fonctions de la date d'enclenchement des transitions.

Définition 2. *Une* contrainte temporelle t-simple *est un triplet $(x, y, dmin)$ composé de deux variables temporelles x et y, et d'une fonction $dmin : \mathbf{d}(x) \times \mathbf{d}(y) \rightarrow \mathbb{R}$ appelée fonction de distance minimale (fonction non nécessairement continue).*

Une contrainte temporelle t-simple $(x, y, dmin)$ est également notée $y - x \geq dmin(x, y)$. La contrainte est satisfaite par un couple de valeurs $(a, b) \in \mathbf{d}(x) \times \mathbf{d}(y)$ si et seulement si $b - a \geq dmin(a, b)$.

Informellement, $dmin(x, y)$ spécifie une distance temporelle minimale entre l'occurrence de l'événement associé à la variable temporelle x et l'occurrence de l'événement associé à la variable temporelle y.

Pour illustrer l'intérêt d'avoir une distance minimale *dmin* dépendant à la fois de x et de y, considérons l'exemple des satellites agiles. Soit x la variable spécifiant la date de fin d'une acquisition *acq*. L'attitude obtenue en terminant *acq* à la date x est notée $Att(x)$, une attitude du satellite étant définie par une direction de pointage et par les vitesses de rotation suivant chacun des trois axes (roulis, tangage, lacet). Soit y la variable donnant la date de début de l'acquisition acq' à réaliser juste après *acq*. L'attitude nécessaire pour commencer acq' à la date y est notée $Att'(y)$. Soit *tminbas* la fonction (disponible dans notre librairie de calculs d'agilité) telle que $tminbas(att, att')$ donne le temps minimal requis pour basculer d'une attitude *att* à une attitude att'. On parle dans ce cas de *rendez-vous en attitude* de *att* vers att'. Alors, en définissant $dmin(x, y) = tminbas(Att(x), Att'(y))$, la contrainte temporelle t-simple $y - x \geq dmin(x, y)$ exprime que la durée entre la fin de *acq* et le début de acq' doit être supérieure à la durée minimale de basculement de l'attitude $Att(x)$ à l'attitude $Att'(y)$.

Dans certains cas, la fonction $dmin(x, y)$ est indépendante de y. Cette remarque concerne notamment le *time-dependent scheduling* [5, 7], dans lequel la durée de réalisation d'une tâche dépend uniquement de la date de début de cette tâche (contrainte temporelle t-simple "dégénérée", de la forme $y - x \geq dmin(x)$ avec $dmin(x)$ le temps de réalisation de la tâche si cette dernière commence à la date x). Les contraintes temporelles t-simples couvrent également les contraintes temporelles simples $y - x \geq c$, en utilisant une fonction de distance minimale constante $dmin = c$.

Remarquons enfin qu'une contrainte temporelle t-simple se réfère à la durée *minimale* d'une transition. Une telle approche est utilisable pour gérer les satellites agiles sous l'hypothèse (réaliste) que tout rendez-vous en attitude faisable en une durée δ est aussi faisable en une durée $\delta' \geq \delta$. Cette hypothèse de faisabilité d'un "rendez-vous paresseux" n'est pas forcément vérifiée pour tout système physique.

2.3 Propriétés de monotonie

Nous commençons par définir la fonction de retard associée à une contrainte temporelle t-simple.

Définition 3. *La* fonction de retard *associée à une contrainte temporelle t-simple* $ct : (x, y, dmin)$ *est la fonction* $delay_{ct} : \mathbf{d}(x) \times \mathbf{d}(y) \to \mathbb{R}$ *définie par :* $delay_{ct}(a, b) = a + dmin(a, b) - b$.

Informellement, $delay_{ct}(a, b)$ donne le retard obtenu en b si l'on enclenche, à la date a, une transition en temps minimal de x vers y. Ce retard s'exprime comme la différence entre la date minimale de fin de la transition $(a + dmin(a, b))$ et la date de fin de transition requise (b). Un retard strictement négatif correspond à une transition se terminant en avance par rapport à la date limite b. Un retard strictement positif correspond à une violation de la contrainte ct. Un retard nul correspond à une arrivée "pile à l'heure".

La fonction de retard peut satisfaire une propriété de monotonie définie ci-dessous, qui sera utilisée par la suite pour identifier des classes traitables.

Définition 4. *Une contrainte temporelle t-simple* $ct : (x, y, dmin)$ *est dite* à retard monotone *si et seulement si sa fonction de retard* $delay_{ct}(.,.)$ *satisfait :* $\forall a, a' \in \mathbf{d}(x)$, $\forall b, b' \in \mathbf{d}(y)$,

$$(a \leq a') \to (delay_{ct}(a, b) \leq delay_{ct}(a', b)) \quad (8)$$
$$(b \leq b') \to (delay_{ct}(a, b) \geq delay_{ct}(a, b')) \quad (9)$$

La contrainte est dite à retard strictement monotone *si les monotonies sur les deux arguments sont strictes.*

La définition 4 signifie que pour être à retard monotone, une contrainte temporelle t-simple $(x, y, dmin)$ doit vérifier d'une part que plus la transition de x vers y est enclenchée tard, plus le retard est grand à l'arrivée en y, et d'autre part que plus l'on cherche à arriver tôt au rendez-vous (en y), plus le retard est élevé.

Nous définissons maintenant les fonctions d'arrivée au plus tôt et de départ au plus tard associées à une contrainte temporelle t-simple. Par la suite, pour une fonction $F : \mathbb{R} \to \mathbb{R}$ et un intervalle fini fermé I de $\mathbb{R}$, on note (1) $firstNeg(F, I)$ le plus petit $a \in I$ tel que $F(a) \leq 0$ (valeur $+\infty$ si une telle valeur n'existe pas) ; (2) $lastNeg(F, I)$ le plus grand $a \in I$ tel que $F(a) \leq 0$ (valeur $-\infty$ si une telle valeur n'existe pas) [1].

Définition 5. *Les fonctions d'arrivée au plus tôt (earliest arrival) et de départ au plus tard (latest departure) associées à une contrainte temporelle t-simple* $ct : (x, y, dmin)$ *sont les fonctions* $earr_{ct}$ *et* $ldep_{ct}$ *définies respectivement sur* $\mathbf{d}(x)$ *et* $\mathbf{d}(y)$ *et données par :*

$$\forall a \in \mathbf{d}(x),\ earr_{ct}(a) = firstNeg(delay_{ct}(a, .), \mathbf{d}(y))$$
$$\forall b \in \mathbf{d}(y),\ ldep_{ct}(b) = lastNeg(delay_{ct}(., b), \mathbf{d}(x))$$

Informellement, $earr_{ct}(a)$ fournit la plus petite date d'arrivée en y sans retard si la transition est enclenchée à la date a. $ldep(b)$ fournit la date d'enclenchement au plus tard pour une arrivée sans retard en b.

1. Les quantités $firstNeg(F, I)$ et $lastNeg(F, I)$ ne sont mathématiquement pas forcément bien définies si la fonction F présente des discontinuités ; nous utilisons en fait implicitement le fait que le travail se fait en machine avec une précision finie.

Pour une contrainte temporelle simple $y - x \geq c$, il est possible de donner une formulation analytique de *earr* et *ldep*. Dans le cas général, il faut passer par le calcul de $\mathit{firstNeg}(F, I)$ et $\mathit{lastNeg}(F, I)$, qui constitue un problème d'optimisation en soi. Une méthode itérative pour approximer $\mathit{firstNeg}(F, I = [a_1, a_2])$ est donnée à l'algorithme 1. Cette méthode généralise la méthode dite de la *fausse position*, utilisable pour trouver un zéro d'une fonction. Appliquée au cas des contraintes temporelles t-simples, elle consiste, si le point $P_1 = (a_1, F(a_1))$ est à retard positif ($F(a_1) > 0$) et le point $P_2 = (a_2, F(a_2))$ est à retard négatif ($F(a_2) < 0$), à étudier comment se comporte le retard $F(a_3)$ en a_3 avec a_3 l'abscisse du point d'intersection de la droite passant par P_1, P_2 avec l'axe des abscisses. Si $P_3 = (a_3, F(a_3))$ est à retard positif (resp. négatif), la recherche se poursuit en prenant $P_1 = P_3$ (resp. $P_2 = P_3$). Si la contrainte temporelle t-simple considérée est à retard strictement monotone, cette méthode converge vers $\mathit{firstNeg}(F, I)$; sinon, la méthode peut renvoyer une valeur $a > \mathit{firstNeg}(F, I)$, mais dans ce cas a satisfait $F(a) \leq 0$ (à la précision près). La vitesse de convergence en pratique est particulièrement bonne pour la fonction de retard des satellites agiles.

Algorithm 1: Méthode possible de calcul de $\mathit{firstNeg}(F, I)$, avec I=$[a_1, a_2]$, maxIter un nombre max d'itérations et prec une précision souhaitée

```
1  firstNeg(F, [a1, a2], maxIter, prec)
2  begin
3      f1 ← F(a1) ; if f1 ≤ 0 then return a1
4      f2 ← F(a2) ; if f2 > 0 then return +∞
5      for i = 1 to maxIter do
6          a3 = (f1 ∗ a2 − f2 ∗ a1)/(f1 − f2)
7          f3 = F(a3)
8          if |f3| < prec then return a3
9          else if f3 > 0 then (a1, f1) ← (a3, f3)
10         else (a2, f2) ← (a3, f3)
11     return a2
```

Sur la base des quantités *earr* et *ldep*, il est possible de définir une seconde propriété de monotonie, plus forte que la monotonie de la fonction de retard (cf. prop. 1) et pouvant être violée en pratique.

Définition 6. *Une contrainte temporelle t-simple ct* : $(x, y, dmin)$ *est dite à décalage monotone si et seulement si elle vérifie,* $\forall a, a' \in \mathbf{d}(x)$, $\forall b, b' \in \mathbf{d}(y)$,

$$(a' \geq a) \rightarrow (earr_{ct}(a') \geq earr_{ct}(a) + (a' - a))$$
$$(b' \geq b) \rightarrow (ldep_{ct}(b') \geq ldep_{ct}(b) + (b' - b))$$

Proposition 1. *Soit ct* : $(x, y, dmin)$ *une contrainte temporelle t-simple. Si ct est à décalage monotone, alors ct est à retard strictement monotone.*

Informellement, une contrainte temporelle t-simple est à décalage monotone si et seulement si d'une part lorsque la date d'enclenchement d'une transition est repoussée, la date d'arrivée au plus tôt est décalée d'autant, et d'autre part lorsque la date de fin de la transition est avancée, la date d'enclenchement au plus tard est décalée d'autant. Dans le cas des satellites agiles, la fonction de distance minimale *dmin* peut présenter des comportements du type de ceux de la figure 1 qui entraînent que les contraintes ne sont pas toujours à décalage monotone.

Les deux propositions suivantes montrent que les propriétés de monotonie sont satisfaites par les contraintes temporelles simples et par plusieurs contraintes temporelles classiquement utilisées dans le time-dependent scheduling (voir [5]).

Proposition 2. *Les contraintes temporelles simples* $y - x \geq c$ *sont à décalage monotone (et donc aussi à retard strictement monotone).*

Proposition 3. *Soit x, y deux variables temporelles correspondant respectivement aux dates de début et de fin d'une tâche. Les résultats de monotonie donnés à la table 1 sont corrects.*

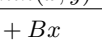
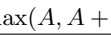
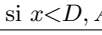
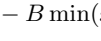
Fonction de distance $dmin(x, y) = dmin(x)$	Forme	*Monotonie*	
		décalage	*retard*
$A + Bx$		oui	oui (strict)
$A - Bx$		non	oui ssi $B \leq 1$ strict ssi $B < 1$
$\max(A, A + B(x - D))$		oui	oui (strict)
A si $x < D$, $A + B$ sinon		oui	oui (strict)
$A - B \min(x, D)$		non	oui ssi $B \leq 1$ strict ssi $B < 1$

TABLE 1 – Monotonie de fonctions de distance utilisées en time-dependent scheduling, avec A, B, D des constantes telles que $A \geq 0$, $B > 0$ et $D > \min(\mathbf{d}(x))$

2.4 Arc-cohérence des contraintes temporelles t-simples

La propriété de retard monotone permet de simplifier l'établissement de l'arc-cohérence.

Proposition 4. *Soit ct* : $(x, y, dmin)$ *une contrainte temporelle t-simple à retard monotone. Etablir l'arc-cohérence aux bornes pour ct est équivalent à établir l'arc-cohérence sur les domaines complets de x et y.*

La raison est qu'en cas de retard monotone, si $\max(\mathbf{d}(x))$ possède un support b sur y, alors b est aussi

un support pour toutes les autres valeurs du domaine de x, et de manière similaire le support de $\min(\mathbf{d}(y))$ sur x est un support pour toutes les valeurs de y.

Proposition 5. *Soit $ct : (x, y, dmin)$ une contrainte temporelle t-simple. L'arc-cohérence aux bornes pour la contrainte ct peut être établie par les modifications de domaine suivantes :*

$$\mathbf{d}(y) \leftarrow \mathbf{d}(y) \cap [earr_{ct}(\min(\mathbf{d}(x))), +\infty[\quad (10)$$
$$\mathbf{d}(x) \leftarrow \mathbf{d}(x) \cap]-\infty, ldep_{ct}(\max(\mathbf{d}(y)))] \quad (11)$$

La règle 10 permet de modifier la date de réalisation au plus tôt associée à y. La règle 11 permet de modifier la date de réalisation au plus tard associée à x. Les modifications de domaines sont telles que les domaines courants $\mathbf{d}(x)$ et $\mathbf{d}(y)$ restent des intervalles fermés.

Proposition 6. *L'arc-cohérence d'une contrainte temporelle t-simple $ct : (x, y, dmin)$ à retard monotone peut être établie par les règles 10 et 11.*

Si l'hypothèse de retard monotone n'est pas satisfaite, l'arc-cohérence aux bornes peut tout de même être établie. Dans ce cas, elle n'implique pas l'arc-cohérence et elle est susceptible de supprimer des valeurs cohérentes des domaines des variables.

2.5 Cadre des TSTN

Sur la base des contraintes temporelles t-simples, il est possible d'introduire un nouveau cadre appelé le cadre des TSTN pour "Time-dependent STN".

Définition 7. *Un TSTN est un couple (V, C) avec V un ensemble fini de variables de domaine $[l, u] \subset \mathbb{R}$ et C un ensemble fini de contraintes temporelles t-simples $(x, y, dmin)$ avec $x, y \in V$.*

Une solution d'un TSTN est une affectation des variables de V qui satisfait toutes les contraintes de C. Un TSTN est dit cohérent s'il admet au moins une solution. Un TSTN est dit à retard monotone (resp. à décalage monotone) si toutes ses contraintes temporelles sont à retard monotone (resp. à décalage monotone).

Exemple Nous reconsidérons l'exemple faisant intervenir trois acquisitions acq_1, acq_2, acq_3. Les durées de transition minimales requises entre acquisitions ne sont cette fois pas considérées comme constantes. Dans le modèle TSTN obtenu, la seule différence est que les contraintes temporelles simples des équations 3 et 4 sont remplacées par les contraintes temporelles t-simples des équations 12 et 13, dans lesquelles pour une acquisition acq_i, $Satt_i(t)$ et $Eatt_i(t)$ donnent respectivement les attitudes requises pour enclencher et finir acq_i à la date t. Le graphe de distance associé à un TSTN peut être défini de manière analogue au graphe de distance associé à un STN (voir la figure 3).

$$sa_1 - ea_3 \geq tminbas(Eatt_3(ea_3), Satt_1(sa_1)) \quad (12)$$
$$sa_2 - ea_1 \geq tminbas(Eatt_1(ea_1), Satt_2(sa_2)) \quad (13)$$

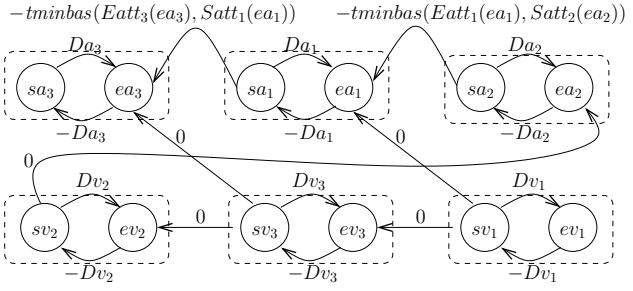


FIGURE 3 – Graphe de distance du TSTN (la référence temporelle x_0 n'est pas représentée)

La proposition suivante étend sur les TSTN un résultat classique sur les STN. Elle est valable que les contraintes soient à retard monotone ou non.

Proposition 7. *Si toutes les contraintes d'un TSTN sont arc-cohérentes aux bornes, alors l'ordonnancement obtenu en fixant des dates au plus tôt (resp. des dates au plus tard) est une solution du TSTN.*

3 Résolution des TSTN

La tâche à laquelle nous nous intéressons est celle de la détermination de la cohérence d'un TSTN et des dates au plus tôt et au plus tard associées à chaque variable temporelle. Les algorithmes proposés reprennent et généralisent des techniques STN existantes.

3.1 Propagation de contraintes

Le premier élément utilisé est un schéma de propagation des bornes min et max des différentes variables temporelles. Cet élément relativement standard s'inspire des approches définies dans [4, 8, 11]. Ces dernières reviennent à maintenir une liste de variables pour lesquelles les contraintes portant sur ces variables doivent être révisées avec, pour chaque variable z de la liste, un maintien de la nature de la (des) révision(s) à effectuer : (a) si z a subi une modification de sa borne min, les bornes min de toutes les variables t liées à z par une contrainte $t - z \geq c$ doivent être révisées ; (b) si z a subi une modification de sa borne max, les bornes max de toutes les variables t liées à z par une contrainte $z - t \geq c$ doivent être révisées.

Par rapport aux approches STN classiques, nous choisissons pour les TSTN une propagation de

contraintes dans laquelle est maintenue une liste contenant non pas des variables mais des contraintes. Cette liste est partitionnée en deux sous-listes, l'une contenant les contraintes à réviser pouvant modifier une borne min (contraintes $y - x \geq dmin(x, y)$ réveillées suite à une modification de $\min x$ et susceptibles de modifier $\min y$), l'autre contenant les contraintes à réviser pouvant modifier une borne max (contraintes $y - x \geq dmin(x, y)$ réveillées suite à une modification de $\max y$ et susceptibles de modifier $\max x$). Par rapport à une version liste de variables, le maintien de listes de contraintes permettra de gérer plus finement certains aspects (voir plus loin).

Enfin, la révision d'une contrainte temporelle t-simple du point de vue des bornes min et/ou max se fait en utilisant les règles de la proposition 6, avec une évaluation de $earr_{ct}$ et $ldep_{ct}$ analytique lorsque cela est possible, par recherche itérative sinon.

3.2 Détection de cycles de longueur négative

Avec des domaines de valeurs bornés, la propagation par arc-cohérence sur les STN est capable de détecter l'incohérence. Cependant, le nombre de révisions de contraintes requises pour arriver à ce résultat est potentiellement prohibitif par rapport à des approches définies dans [3, 4], qui utilisent le fait que l'incohérence d'un STN est équivalente à l'existence d'un cycle de longueur négative dans son graphe de distance.

L'idée de base de ces approches consiste à détecter de tels cycles à la volée en maintenant des *chaînes de propagation*. Ces dernières correspondent à des explications des valeurs courantes des bornes min et max des différentes variables. Une contrainte $y - x \geq c$ est dite active du point de vue des bornes min (resp. des bornes max) si et seulement si la dernière révision de cette contrainte est responsable de la dernière modification de la borne min de y (resp. de la borne max de x). [3] montre que s'il existe un cycle dans le graphe orienté dont les arcs correspondent aux contraintes actives du point de vue des bornes min, alors le STN est incohérent. L'intuition est que si un cycle de propagation $x_1 \to x_2 \to \ldots \to x_n \to x_1$ est détecté, alors la valeur minimale de x_1 modifie la valeur minimale de x_2... qui elle-même modifie la valeur minimale de x_n qui elle-même modifie la valeur minimale de x_1, et donc il est possible de vider le domaine de x_1 en parcourant ce cycle un nombre suffisant de fois. Le même résultat s'applique au graphe orienté contenant un arc par contrainte active du point de vue des bornes max.

Le point bloquant pour adapter ces techniques au cas des contraintes temporelles t-simples est que pour un TSTN dans le cas général, l'existence d'un cycle de propagation n'est pas nécessairement synonyme d'incohérence, comme le montre l'exemple suivant.

Exemple Soit $(V, C) = (\{x, y\}, \{ct_1, ct_2\})$ un TSTN contenant deux variables temporelles de domaines $\mathbf{d}(x) = \mathbf{d}(y) = [0.5, 2]$, et deux contraintes $ct_1 : y - x \leq 0.5$ et $ct_2 : y - x \geq dmin(x, y)$, avec $dmin$ la fonction de distance minimale définie par $dmin(a, b) = 1 - a/2$. La fonction de retard associée à ct_2 est strictement monotone (elle vaut $delay_{ct_2}(a, b) = 1 + a/2 - b$).

L'établissement de l'arc-cohérence pour ct_2 (règle 10) modifie la borne min de y et donne $\mathbf{d}(y) = [1 + 1/4, 2]$. Une propagation sur la contrainte ct_1 modifie alors la borne min de x et donne $\mathbf{d}(x) = [1 - 1/4, 2]$. Le résultat obtenu est un cycle de propagation, puisque la borne min de x a modifié la borne min de y qui elle-même a modifié la borne min de x. Dans le cadre STN classique, l'existence d'un tel cycle implique une incohérence. Dans le cadre TSTN, une telle conclusion est fausse puisque par exemple l'instanciation $x = 1$, $y = 1.5$ est cohérente.

La raison à cela est que dans le cadre TSTN, les réductions de domaines induites par des reparcours d'un cycle de propagation peuvent devenir de plus en plus petites. C'est ce qui se produit sur l'exemple ci-dessus où après n parcours du cycle on a $\mathbf{d}(x) = [1 - 1/2^n, 2]$. D'un point de vue implémentation en machine, la précision finie implique que le parcours des cycles s'arrête à un moment donné, mais potentiellement après un très grand nombre d'itérations.

L'exemple montre également que la monotonie stricte de la fonction de retard ne suffit pas pour conclure à l'incohérence en cas de détection de cycle. Une condition suffisante, satisfaite pour les STN classiques, est donnée ci-dessous. Cette condition assure qu'un cycle ne devient pas "moins négatif" au fur et à mesure de ses reparcours.

Proposition 8. *Si un cycle de propagation est détecté dans un TSTN, alors il suffit que toutes les contraintes temporelles t-simples du cycle soient à décalage monotone pour conclure que le TSTN est incohérent.*

Proposition 9. *En particulier, (1) pour un TSTN à décalage monotone, l'existence d'un cycle de propagation implique l'incohérence du TSTN ; (2) pour un TSTN dont le graphe de distance ne contient aucun cycle impliquant une contrainte temporelle t-simple à décalage non monotone, l'existence d'un cycle de propagation implique l'incohérence du TSTN.*

Dans l'application satellite agile qui motive ce travail, la fonction de distance minimale n'est pas monotone, mais par contre le point 2 de la proposition 9 s'applique sur les cas d'étude traités. Déduire une incohérence suite à une détection de cycle est dans ce cas correct. Si aucune des conditions de la proposition 9 ne s'applique, plusieurs options sont envisageables. La

première consiste à ne pas inclure les contraintes temporelles à décalage non monotones dans les chaînes de propagation. La seconde, incorrecte dans le sens où elle peut conduire à des détections d'incohérence à tort, consiste à considérer que le TSTN est incohérent dès qu'un cycle de propagation est détecté.

En termes de complexité, la proposition suivante généralise aux TSTN, et donc aux problèmes de time-dependent scheduling, les résultats de complexité polynomiale disponibles sur les STN.

Proposition 10. *L'algorithme utilisant propagation par arc-cohérence aux bornes et détection de cycle sur un TSTN (V, C) établit l'arc-cohérence aux bornes en $O(|V||C|)$ révisions de contraintes (borne indépendante de la taille des domaines des variables).*

Côté implémentation, la détection de cycles de propagation à la volée s'appuie sur une structure de donnée efficace [1] utilisée pour maintenir des ordres topologiques dans les graphes de propagation des bornes min et max. L'existence de cycles dans ces graphes est en effet équivalente à la non existence d'ordres topologiques des nœuds dans ces mêmes graphes.

3.3 Dépropagation de contraintes

Les méthodes de propagation de contraintes permettent directement de traiter le cas de l'ajout d'une contrainte au TSTN de départ ou le cas du renforcement d'une contrainte. Concernant le retrait ou l'assouplissement d'une contrainte, il est possible de réutiliser directement les stratégies définies dans [11] pour les STN. Ces stratégies permettent de rétablir à moindre coût les bornes min et max des variables temporelles. La technique de base consiste à suivre les chaînes de propagation pour savoir quels domaines de variables réinitialiser et quelles contraintes repropager. Lors du retrait ou d'un assouplissement d'une contrainte $y - x \geq dmin(x, y)$, si cette contrainte est active du point de vue de la borne min de y (resp. de la borne max de x), alors la borne min de y (resp. la borne max de x) est réinitialisée à la valeur qu'elle avait avant toute propagation. Cette réinitialisation peut elle-même déclencher d'autres réinitialisations. Les contraintes du TSTN de la forme $y - z \geq dmin(z, y)$ (resp. $z - x \geq dmin(x, z)$) sont ajoutées à la liste des contraintes à propager du point de vue des bornes min (resp. max) des variables.

La seule différence par rapport à la littérature STN est l'utilisation de listes de contraintes à propager (et non de variables), qui permet de faire une dépropagation légèrement plus fine : sur l'exemple de réinitialisation de la borne min de y, la version classique STN ajoute à une liste des variables à propager toute variable z liée à y par une contrainte $y - z \geq dmin(z, y)$, et ce faisant repropage *in fine* toutes les contraintes de la forme $u - z \geq dmin(z, u)$, même celles avec $u \neq y$.

3.4 Ordonnancement des révisions de contraintes

Nous utilisons une dernière technique dont le but est de minimiser le nombre de révisions de contraintes. Cet objectif est intéressant pour les STN mais l'est d'autant plus pour les TSTN, pour lesquels le coût d'une révision de contrainte peut être significativement plus élevé. L'approche proposée reprend et étend une technique développée pour les STN$^-$[8], une sous-classe des STN dans laquelle les contraintes temporelles simples doivent pouvoir être mises sous la forme $y - x \geq c$ avec $c \geq 0$. L'idée consiste à construire les composantes fortement connexes du graphe de distance, à les ordonner suivant un ordre topologique, et à utiliser cet ordre topologique pour savoir dans quel ordre propager les contraintes. Nous rappelons tout d'abord la définition de la notion de composante fortement connexe.

Définition 8. *Soit $G = (V, A)$ un graphe orienté, avec V l'ensemble des nœuds et A l'ensemble des arcs. Une composante fortement connexe de G (SCC,* Strong Connected Component*) est un sous-graphe G' maximal de G tel qu'il existe dans G' un chemin de n'importe quel nœud à n'importe quel autre nœud.*

*Le DAG (*Directed Acyclic Graph*) des SCCs de G est le graphe orienté dont les nœuds sont les SCCs de G et tel qu'il existe un arc d'un SCC c_1 vers un SCC c_2 si et seulement si il existe dans G un arc d'un des sommets de c_1 vers un des sommets de c_2.*

Un ordre topologique des SCCs est un ordre $\preceq$ tel que chaque SCC c est placé dans cet ordre strictement après chacun de ses parents c' dans le DAG des SCCs ($c' \prec c$). Pour un nœud x du graphe de départ G, on note $scc(x)$ le SCC qui contient x.

Propager les contraintes temporelles en suivant un ordre topologique des SCCs du graphe de distance revient à utiliser le résultat selon lequel résoudre un problème de plus court chemin dans un graphe acyclique est plus facile que dans un graphe quelconque. Pour appliquer ce résultat, les contraintes à propager sont ordonnées suivant un ordre topologique des SCCs. Plus précisément, concernant la propagation des bornes min, on propage en priorité les contraintes $y - x \geq dmin(x, y)$ avec $scc(y)$ maximal au sens de l'ordre des SCCs, et en cas d'égalité en privilégiant les contraintes telles que $scc(x) \neq scc(y)$, pour repousser au maximum la propagation des contraintes internes à un SCC. Pour la propagation des bornes max, les contraintes sont ordonnées par $scc(x)$ croissant et en cas d'égalité en privilégiant les contraintes telles que $scc(y) \neq scc(x)$. Sur l'exemple de la figure 3, les SCCs sont représentés en pointillés. Un mauvais ordre de

propagation des bornes min consisterait à propager d'abord la contrainte entre sa_2 et ea_1 puis celle entre sa_1 et ea_3. Un bon ordre, cohérent avec l'ordre des SCCs, consisterait à faire l'inverse.

Par rapport à l'utilisation des SCCs faite dans [8] pour les STN$^-$, la méthode proposée est adaptée non seulement aux STN généraux mais aussi aux TSTN. Côté implémentation, pour ne pas recalculer le DAG des SCCs après chaque ajout ou retrait de contrainte, nous utilisons des algorithmes adaptés au maintien dynamique des SCCs [9, 17].

4 Expérimentations

Les techniques proposées précédemment sont implémentées dans un outil d'ordonnancement fonctionnant à base de recherche locale. Les mouvements locaux consistent en des modifications d'un ordonnancement courant. Ils se traduisent par des ajouts ou des retraits de contraintes temporelles t-simples. Le côté recherche locale implique qu'il est rapide de faire/défaire un mouvement local, dans un esprit similaire aux outils Comet [10] et LocalSolver [2]. Une différence par rapport aux outils classiques de recherche locale à base de contraintes est que dans le cas des TSTN, les variables temporelles ne sont pas instanciées à une valeur unique (on maintient pour chacune un intervalle de valeurs). L'outil de résolution STN/TSTN est implémenté en Java. Les résultats sont obtenus sur un processeur Intel i5-520 1.2GHz, 4GBRAM.

Des expérimentations non détaillées ici ont tout d'abord été réalisées sur des STN obtenus à partir de problèmes d'ordonnancement de la SMT-LIB. L'objectif était de valider l'intérêt de l'heuristique de propagation basée sur un ordre topologique des SCCs. Cette heuristique se révèle être une stratégie robuste permettant, sur certains problèmes, de gagner un ou plusieurs ordres de grandeur en termes de nombre de révisions de contraintes. Plus précisément, sur des STN cohérents, cette heuristique est toujours au moins aussi bonne qu'une approche par gestion des contraintes à propager sous forme de liste FIFO ou LIFO. Sur des STN incohérents, propager suivant l'ordre des SCCs n'est par contre pas toujours la stratégie qui établit le plus vite l'incohérence.

Nous détaillons ci-après les expérimentations réalisées sur les TSTN dans le contexte satellites agiles. Le problème considéré est une simple contrainte de non chevauchement sur une séquence ordonnée de n acquisitions $acq_1 \rightarrow \ldots \rightarrow acq_n$, avec n variant entre 5 et 13. Ces acquisitions correspondent à des zones au sol situées entre le nord le l'Espagne et le nord de la France. La contrainte de non chevauchement entre acquisitions s'exprime comme un ensemble de contraintes temporelles t-simples de la forme $s_{i+1} - e_i \geq tminbas(Eatt_i(e_i), Satt_{i+1}(s_{i+1}))$ avec, pour une acquisition j, s_j/e_j les dates de début/fin de cette acquisition, et $Satt_j(t)/Eatt_j(t)$ les attitudes nécessaires pour enclencher/finir j à la date t. Sont ajoutées à cela des contraintes temporelles simples fixant une durée constante pour chaque prise de vue.

Deux méthodes sont comparées : d'un côté une approche TSTN avec prise en compte des temps de transition exacts entre acquisitions, et de l'autre une approche STN avec précalcul de majorants sur les temps de transition. Les ordonnancements obtenus dans les deux cas sont flexibles dans le sens où la propagation sur les STN/TSTN fournit en sortie des variables temporelles dont le domaine élagué n'est en général pas réduit à une seule valeur possible. Le critère étudié est la flexibilité temporelle moyenne, mesurée comme la moyenne sur l'ensemble des variables temporelles x de la différence entre date au plus tôt et date au plus tard pour x. La flexibilité temporelle est importante pour offrir le plus de latitude possible concernant le choix de l'angle de réalisation d'une prise de vue, qui influence la qualité image.

Trois scénarios sont étudiés. Dans le premier scénario, les acquisitions sont des bandes de longueur 80km environ, à observer avec un cap de 0 degré, le cap correspondant à l'angle entre la trace au sol du satellite et la direction de balayage de la zone. La figure 4(a) montre que dans ce cas la flexibilité temporelle obtenue avec les TSTN est à peine meilleure que celle obtenue avec des STN. La raison est que si toutes les prises de vue sont réalisées avec un cap de 0 degré, les temps de transition minimaux entre prises de vue varient peu avec les dates d'enclenchement des transitions.

Dans le deuxième scénario, le cap de balayage des zones passe à 30 degrés. La figure 4(b) montre que la flexibilité temporelle obtenue avec les TSTN est meilleure d'environ 20 secondes en moyenne, et que la différence de flexibilité entre STN et TSTN croît avec le nombre d'acquisitions planifiées.

Dans le troisième et dernier scénario, la longueur des zones à balayer passe à 40km environ et le cap de balayage est choisi aléatoirement pour chaque zone. Dans ce cas, l'approche STN ne permet de planifier que les séquences de longueur $n = 5$ et 6. Elle conclue à une incohérence du problème pour $n \geq 7$. A l'inverse, l'approche TSTN permet de planifier l'ensemble des 13 acquisitions. Une des raisons est que plus les caps de balayage sont distincts, plus les temps de transition minimaux entre prises de vue dépendent des dates d'enclenchement des transitions. La possibilité d'avoir des caps de balayage distincts est importante en pratique. Elle permet est effet, étant donné un polygone au sol, de le découper suivant des bandes dont l'orien-

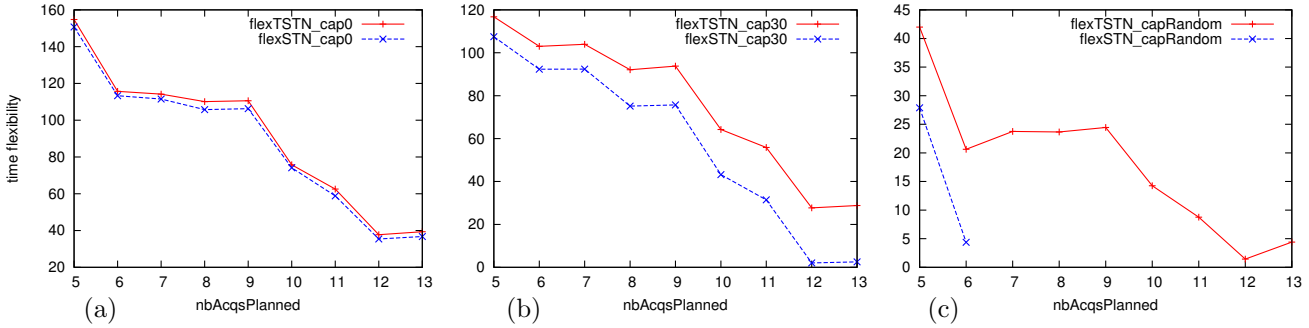


FIGURE 4 – Comparaison des flexibilités temporelles moyennes, en secondes, obtenues avec l'utilisation de majorants précalculés sur les temps de transition (flexSTN) et avec les temps de transitions exacts (flexTSTN)

tation est libre, et donc potentiellement de réduire le nombre de bandes à balayer.

Pour donner un ordre d'idée des temps de calcul, pour 13 acquisitions ajoutées une à une au plan courant, une précision de une seconde sur les dates, et un nombre d'itérations égal à 10^4 pour la convergence des fonctions *firstNeg* et *lastNeg*, l'approche TSTN consomme en moyenne 2ms par ajout. Avec l'approche STN, le temps de calcul est inférieur à 0.1ms par ajout. Pour des précisions de 10^{-1}, 10^{-2}, 10^{-3} et 10^{-4} seconde sur les dates, les temps de calcul avec les TSTN passent respectivement à 3ms, 12ms, 66ms et 182ms par ajout. Une approche type peut consister à étudier d'abord les différents ordonnancements possibles d'un point de vue gros grain puis à préciser les choses en utilisant une précision plus fine.

En conclusion, ce papier a présenté les TSTN, leurs propriétés, des techniques de résolution et une application à la gestion de satellites agiles. Il serait intéressant d'étudier d'autres propriétés de ce cadre et de tester l'approche sur d'autres applications.

Références

[1] M. A. Bender, R. Cole, E. D. Demaine, M. Farach-Colton, and J. Zito. Two simplified algorithms for maintaining order in a list. *Proc. of ESA-02*, pages 152–164, 2002.

[2] T. Benoist, B. Estellon, F. Gardi, R. Megel, and K. Nouioua. Localsolver 1.x : a black-box local-search solver for 0-1 programming. *4OR*, 9(3) :299–316, 2011.

[3] R. Cervoni, A. Cesta, and A. Oddi. Managing dynamic temporal constraint networks. *Proc. of AIPS-94*, pages 13–18, 1994.

[4] A Cesta and A Oddi. Gaining efficiency and flexibility in the simple temporal problem. *Proc. of TIME-96*, pages 45–50, 1996.

[5] T. Cheng, Q. Ding, and B. Lin. A concise survey of scheduling with time-dependent processing times. *EJOR*, 152 :1–13, 2004.

[6] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49 :61–95, 1991.

[7] S. Gawiejnowicz. *Time-Dependent Scheduling*. Springer, 2008.

[8] A. Gerevini, A. Perini, and F. Ricci. Incremental algorithms for managing temporal constraints. *Proceedings of ICTAI-96*, pages 360–365, 1996.

[9] B. Haeupler, T. Kavitha, R. Mathew, S. Sen, and R. E. Tarjan. Incremental cycle detection, topological ordering, and strong component maintenance. *ACM Transactions on Algorithms*, 8(1), 2012.

[10] P. Van Hentenryck and L. Michel. *Constraint-Based Local Search*. The MIT Press, 2005.

[11] I. Shu, R. Effinger, and B. Williams. Enabling fast flexible planning through incremental temporal reasoning with conflict extraction. *Proc. of ICAPS-05*, pages 252–261, 2005.

[12] M. Lemaître, G. Verfaillie, F. Jouhaud, J.-M. Lachiver, and N. Bataille. Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology*, 6 :367–381, 2002.

[13] U. Montanari. Networks of constraints : fundamental properties and applications to picture processing. *Information Sciences*, 7(2) :95–132, 1974.

[14] L. Planken, M. de Weerdt, and N. Yorke-Smith. Incrementally solving STNs by enforcing partial path consistency. *Proc. of ICAPS-10*, pages 129–136, 2010.

[15] L. R. Planken, M. M. de Weerdt, and R. P.J. van der Krogt. P3C : A new algorithm for the simple temporal problem. *Proc. of ICAPS-08*, pages 256–263, 2008.

[16] Challenge ROADEF'03. http://challenge.roadef.org.

[17] L. Roditty and U. Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM Journal on Computing*, 37(5) :1455–1471, 2008.

[18] K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120 :81–117, 2000.

[19] L. Xu and B. Y. Choueiry. A new efficient algorithm for solving the simple temporal problem. *Proc. of TIME-ICTL-03*, pages 210–220, 2003.

La Résolution étendue étroite

Nicolas Prcovic
LSIS - Université d'Aix-Marseille
nicolas.prcovic@lsis.org

Résumé

La résolution étendue (ER) (c'est-à-dire, la résolution incorporant la règle d'extension) est un système de preuve plus puissant que la résolution standard (Res) car elle permet de résoudre en temps polynômial certaines classes d'instances que Res ne peut traiter qu'en temps exponentiel. Cependant, elle est très difficile à mettre en pratique car la règle d'extension accroit considérablement la taille de l'espace de recherche de la preuve. On dit qu'une résolution est étroite si l'application de la règle de résolution produit une résolvante contenant au maximum trois littéraux. Dans cet article nous présentons deux variantes de ER : la résolution étendue étroite (ER3) et la résolution à refragmentation. Ces deux systèmes de preuves p-simulent ER : toute preuve de l'une n'est que polynômialement plus longue que celle de l'autre. ER3 consiste simplement à n'autoriser que des résolutions étroites dans ER. Ceci permet une diminution exponentielle de l'espace de recherche d'une preuve dans ER. La résolution à refragmentation est une variante de ER3 qui permet une intégration facile de la résolution étendue dans n'importe quel solveur appliquant la résolution.

Abstract

Extended Resolution (ie, Resolution incorporating the extension rule) is a more powerful proof system than Resolution because it can find polynômially bounded refutations of some SAT instances where Resolution alone cannot (and at the same time, every proof with resolution is still a valid proof with extended resolution). However it is very difficult to put it into practice because the extension rule is an additionnal source of combinatorial explosion, which tends to lengthen the time to discover a refutation. We call a restriction of Resolution forbiding the production of resolvents of size greater than 3 *Narrow Resolution*. We show that Narrow Extended Resolution p-simulates (unrestricted) Extended Resolution. We thus obtain a proof system whose combinatorics is highly reduced but which is still as powerful as before. We also define *Splitting Resolution*, a way to integrate easily Narrow Extended Resolution into any resolution-based solver

1 Introduction

Les systèmes de preuve basés sur la résolution sont utilisés pour permettre de trouver la réfutation d'une formule booléenne supposée insatisfiable. Cependant, certaines familles d'instances [**?**, **?**, **?**] nécessitent de produire un nombre exponentiel de résolvantes. La résolution étendue [**?**] ajoute une régle supplémentaire à la résolution, consistant à ajouter les trois clauses correspondant à $x \Leftrightarrow l_1 \vee l_2$ à la formule, où x est une nouvelle variable. Dans ce cadre, Cook [**?**] exhibe une réfutation de longueur polynômiale du problème des pigeons. Personne n'a jamais trouvé de classe d'instances nécessitant la production d'un nombre exponentiel de résolvantes avec la résolution étendue. Mais, bien que rendant un système de preuve par résolution plus puissant, la règle d'extension introduit une source supplémentaire d'explosion combinatoire. En effet, le nombre potentiel de résolvantes que l'on peut produire n'est plus une exponentielle du nombre de variables de la formule mais du nombre total de variables, qui inclut les variables ajoutées par les extensions. Or, le nombre d'extensions que l'on peut faire n'est pas nécessairement polynômial. C'est pourquoi les rares systèmes l'incluant (GUNSAT [**?**], GlucosEr [**?**], ECL [**?**]) restreignent beaucoup l'application de cette règle afin que son avantage, le possible raccourcissement de la longueur de la réfutation, ne soit pas contrebalancé par son inconvénient, le traitement des clauses supplémentaires.

Rappelons qu'on dit qu'un système de preuve P *p-simule* un système de preuve Q si toute réfutation produite par P n'est que polynômialement plus longue que celle produite par Q. Dans cet article, nous proposons deux variantes de la Résolution étendue, qui la p-simulent et sont destinés à améliorer son efficacité pratique. La première variante, appelée Résolution étendue étroite, se contente de limiter l'application de la règle de résolution à la production de clauses de

taille 3 au maximum. La combinatoire de ce système de preuve se trouve dont très fortement réduit. La seconde variante, appelée Résolution à refragmentation, remplace la règle d'extension de la Résolution étendue étroite par une règle qui combine une résolution et une extension pour produire deux nouvelles clauses de taille 3 à partir de deux clauses de taille 3. Elle permet ainsi de s'intégrer facilement à n'importe quel solveur en substituant la nouvelle règle à la règle de résolution.

Après avoir rappelé quelques notions sur le problème SAT, donné les rapports entre longueur et largeur de preuve et introduit la Résolution étendue (section 2), nous démontrons que la Résolution étendue étroite p-simule la Résolution étendue (section 3) puis que c'est aussi le cas pour la Résolution à refragmentation (section 4).

2 Notions préliminaires

Une *formule booléenne*, sous sa forme dite *CNF*, est une conjonction de clauses. Une clause est une disjonction de littéraux. Un littéral est soit une variable booléenne, soit sa négation. On définit var(l) comme étant la variable du littéral l. Les variables peuvent être affectées à la valeur vrai ou faux. Une clause peut être représentée par l'ensemble de ses littéraux. Une formule SAT peut être représentée par l'ensemble de ses clauses. Un *modèle* d'une formule ϕ est une affectation de variables qui est telle que ϕ est vraie. Une formule n'ayant pas de modèle est dite *insatisfiable*. La clause vide, notée $\square$, est toujours fausse et si une formule la contient alors elle est insatisfiable.

2.1 La résolution

Les systèmes formels de preuve propositionnelle permettent de dériver d'autres formules à partir d'une formule ϕ, grâce à des règles d'inférence. En particulier, le système de Robinson [**?**], que nous appellerons `Res`, permet de dériver des clauses induites à partir des clauses d'une formule ϕ grâce à la règle de *résolution* :

$$\frac{C_1 \qquad C_2}{C_1 \backslash \{l\} \cup C_2 \backslash \{\bar{l}\}}$$

La clause inférée par résolution de C_1 avec C_2 sera appelé *résolvante* sur var(l) de C_1 et C_2. L'intérêt de `Res` est qu'il permet d'établir des réfutations de formules (ie, des preuves de leur insatisfiabilité) dans la mesure où une formule est insatisfiable si et seulement si il existe une suite de résolutions qui dérive la clause vide. `Res` fonctionne par saturation de l'application de sa règle : si une formule est satisfiable, `Res` le détecte quand plus aucune résolvante non redondante ne peut être dérivée.

Une dérivation est une suite d'applications de la règle de résolution permettant de dériver une clause. Elle constitue la preuve que cette clause est une conséquence logique de la formule. Elle peut se représenter grâce à un arbre binaire dont les feuilles sont des clauses de ϕ, les nœuds internes sont des résolvantes et la racine est la clause dérivée. Une dérivation de la clause vide s'appelle une *réfutation*.

2.2 Longueur et largeur de preuve

On appelle *longueur d'une preuve*, le nombre de résolvantes qu'elle contient. On appelle *taille* d'une clause le nombre de littéraux qu'elle contient. On appelle *largeur d'une preuve* la taille de la plus grande clause apparaissant dans cette preuve. Nous dirons qu'une résolution est *étroite* quand la résolvante qu'elle produit est de taille 3 au maximum.

Les relations entre longueur et largeur de preuve sont fortes. Il est clair qu'une preuve étroite est forcément courte. Plus précisément, si la largeur de la preuve est k alors la preuve ne peut contenir que de l'ordre de $\Theta(n^k)$ résolvantes différentes, où n est le nombre de variables de la formule. Si k est une constante alors la longueur de la preuve est polynômialement bornée. Dans [**?**], les auteurs montrent complémentairement que lorsqu'on peut produire une preuve courte, on peut faire en sorte qu'elle soit étroite. A partir de cette relation entre largeur et longueur de preuve, il est possible de définir des algorithmes de recherche de réfutations qui restreignent la largeur des résolvantes produites. Par exemple, le pré-traitement de Satz[**?**] (avant une résolution complète à la DPLL) consiste à effectuer toutes les résolutions étroites possibles jusqu'à saturation (ou dérivation éventuelle de la clause vide). Plus généralement, une manière de garantir une complexité polynômiale consiste à restreindre la résolution à la production de résolvantes de taille inférieure à une borne donnée. Evidemment, cela se fait au détriment de la complétude dans la mesure où il est parfois nécessaire de dériver des grandes clauses avant d'obtenir la clause vide. On peut aussi définir une méthode complète qui restreint la résolution à la dérivation de résolvantes de taille k maximum mais qui incrémente la valeur de k à chaque fois qu'il y a saturation [**?**]. Dans cet article, nous explorons une autre voie. Au lieu d'augmenter la valeur de k, que nous fixons une fois pour toute à 3, nous nous plaçons dans la cadre de la résolution étendue, qui va nous permettre de rajouter des clauses permettant de n'effectuer que des résolutions étroites.

2.3 La résolution étendue

La résolution étendue [**?**] consiste à ajouter à `Res` la *règle d'extension* en plus de la règle de résolution. Nous appellerons `ER` ce nouveau système de preuve. En toute généralité, la règle d'extension permet d'ajouter (à l'ensemble des clauses) les clauses correspondant à la formule $x \Leftrightarrow F$, où x est une nouvelle variable et F est n'importe quelle formule portant sur des variables existant déjà. A partir de maintenant, nous distinguerons les nouvelles variables introduites par des extensions, dites *variables d'extension*, des variables de la formule initiale, dites *variables d'entrée*.

On peut toujours se restreindre à ce que F soit uniquement du type $l_1 \vee l_2$ où l_1 et l_2 sont des littéraux portant sur des variables déjà présentes. En effet, il est facile de décomposer récursivement n'importe quelle formule F en une composition de disjonctions de sous-formules, éventuellement négatives. En associant chaque sous-formule à une nouvelle variable, on obtient un ensemble d'extensions de type $x_i \Leftrightarrow (l_1 \vee l_2)$ au lieu d'une seule du type $x \Leftrightarrow F$. Ce type d'extension portant sur la disjonction de deux littéraux préexistants sera appelée *extension binaire disjonctive*. Une extension $x \Leftrightarrow (l_1 \vee l_2)$ consiste à ajouter les clauses $\overline{x} \vee l_1 \vee l_2$, $x \vee \overline{l_1}$ et $x \vee \overline{l_2}$. Il faut noter qu'il est aussi parfaitement possible de se restreindre à des extensions binaires conjonctives (i.e., de type $x \Leftrightarrow (l_1 \wedge l_2)$, qui consiste à ajouter les clauses $x \vee \overline{l_1} \vee \overline{l_2}$, $\overline{x} \vee l_1$ et $\overline{x} \vee l_2$).

L'intérêt de rajouter la règle d'extension est qu'elle rend le système plus puissant dans la mesure où certains problèmes dont la preuve est de longueur nécessairement exponentielle dans `Res` ont une preuve polynômiale dans `ER`. C'est le cas du fameux problème des pigeons [**?**, **?**]. En dehors de ce problème particulier, on peut identifier des cas plus généraux où une extension permet un raccourcissement de preuve. Considérons une preuve contenant des séquences de résolutions qui font que $C_1 \vee C$ est utilisée pour dériver $C_1 \vee C'$ et que $C_2 \vee C$ est utilisée pour dériver $C_2 \vee C'$ grâce à la même suite de clauses qui permet de passer de C à C' (C_1, C_2, C et C' sont des clauses). On peut vérifier que grâce aux clauses émanant de $x \Leftrightarrow C_1 \wedge C_2$, on dérive d'abord $x \vee C$ à partir de $C_1 \vee C$ et $C_2 \vee C$, puis on effectue *une seule fois* la séquence de résolutions sur C permettant de dériver $x \vee C'$, à partir de quoi on peut à nouveau dériver $C_1 \vee C'$ et $C_2 \vee C$ (encore grâce aux clauses émanant de la même extension). Cette idée, appelée *compression de preuve* dans [**?**] est mise en œuvre lors de la phase d'apprentissage d'une clause dans un solveur CDCL. Par contre, dans [**?**], quand une clause $\alpha \vee \beta$, où α et β sont des sous-clauses de longueur ≥ 2, est apprise par un solveur CDCL, l'extension $x \Leftrightarrow \alpha$ est ajoutée. Dans les expérimentations de cet article, α ne contient que deux littéraux, ce qui revient donc à effectuer une extension binaire disjonctive. C'est aussi ce type d'extension que nous utiliserons ici.

3 La Résolution étendue étroite

Bien que `ER` soit théoriquement plus puissant que `Res`, i.e. permet de trouver des preuves plus courtes, en pratique elle pose des difficultés qui l'ont empêché jusqu'à récemment d'être performante. Le choix des extensions à effectuer est une question difficile. Personne n'y a trouvé de réponse satisfaisante depuis plus de quarante ans. Et quand bien même les bons choix seraient faits, les résultats ne seraient pas automatiquement meilleurs. Par exemple, quand nous avons ajouté les $\Theta(n^3)$ clauses issues de la règle d'extension proposés par Cook aux $\Theta(n^2)$ clauses du problème des n pigeons, nous avons expérimenté qu'aucun type de solveurs (qu'il soit de type CDCL ou DPLL, que ce soit DP60 [**?**] ou de la SL-résolution [**?**]) ne résolvait plus rapidement (bien au contraire) le problème alors que son temps de résolution devenait théoriquement polynômial. A moins d'avoir un bon moyen de choisir les résolvantes à produire, il y a le risque de produire à peu près les mêmes résolvantes qu'auparavant et beaucoup d'autres encore.

Avec `Res`, la taille maximale d'une résolvante est n, où n est le nombre de variables de la formule. Le nombre de résolvantes possible est en $\Theta(3^n)$ car, quelle que soit la variable, elle peut apparaître dans une clause positivement, négativement ou ne pas apparaître. Dans `ER`, le nombre de résolvantes possible est en $\Theta(3^{(n+n')})$ où n' est le nombre d'extensions. Il faut remarquer que n' peut être lui-même une exponentielle de n, étant donné que chaque extension correspond à l'une des formules possibles que l'on peut construire à partir de n variables. L'espace de recherche de la preuve dans `ER` est donc exponentiellement plus grand que dans `Res`. C'est pourquoi les algorithmes voulant intégrer la règle d'extension pour essayer de produire des réfutations plus courtes ont cherché jusqu'à présent à limiter fortement le nombre d'extensions possibles. Or, dans [**?**], nous avions pris le parti inverse : autoriser a priori toutes les extensions possibles mais limiter l'application de la règle de résolution à des résolutions étroites (ie, la taille des résolvantes ne doit pas dépasser 3). Nous obtenions donc un nouveau système de preuve, appelé *Résolution étendue étroite* (`ER3`), qui était démontré complet et tel que toute preuve arborescente dans `ER` pouvait se réduire en temps polynômial en preuve dans `ER3` dans [**?**]. La question de savoir si `ER3` p-simule `ER` était laissée ouverte. La démonstration de la réponse positive (et simple, du moment que nous avons maintenant pris connaissance d'un résultat

qui nous avait échappé) à cette question est donnée plus bas dans le présent article.

L'intérêt pratique de ER3, à partir du moment où toute résolvante possède une taille limitée à 3, est que l'espace de recherche d'une preuve se trouve réduite à $\theta((n+n')^3)$. Il s'agit au pire d'une exponentielle de n, comme pour Res. Nous nous retrouvons donc à chercher des preuves potentiellement plus courte que dans Res dans un espace de recherche dont l'ordre de grandeur est à peu près du même ordre que celui de Res.

La Résolution étendue étroite ne peut s'appliquer que si les formules CNF que nous cherchons à réfuter sont des instances 3-SAT (i.e., ne contiennent que des clauses de taille 3). Dans le cas contraire, il est toujours possible de ramener une formule CNF à une instance 3-SAT équivalente du point de vue de la satisfiabilité. La technique standard consiste à remplacer une clause $C = a_1 \vee a_2 \vee ... \vee a_k$ par $k-2$ clauses ternaires $a_1 \vee a_2 \vee \overline{x_2}$, $x_2 \vee a_3 \vee \overline{x_3}$, ..., $x_{k-3} \vee a_{k-2} \vee \overline{x_{k-2}}$ et $x_{k-2} \vee a_{k-1} \vee a_k$, où les $k-3$ variables x_i sont nouvelles. Cela se fait simplement en ajoutant les extensions $x_2 \Leftrightarrow (a_1 \vee a_2)$ et $\forall i, 3 \leq i < k-1, x_i \Leftrightarrow (a_{i-1} \vee x_{i-1})$. En effet, il suffit de remarquer que les clauses ternaires ainsi ajoutées sont celles du type $x_i \vee a_{i+1} \vee \overline{x_{i+1}}$ qui remplacent C, tandis qu'en appliquant linéairement la suite de $2k-4$ résolutions entre les clauses binaires ajoutées et C, on dérive la dernière clause de remplacement $x_{k-2} \vee a_{k-1} \vee a_k$.

On peut obtenir une preuve arborescente de largeur 3 à partir d'une preuve arborescente de largeur quelconque par une suite d'*amincissements*. Un amincissement de preuve consiste à réduire d'au moins une unité le nombre de ses résolvantes de taille 4. Un amincissement se fait grâce à une extension $x \Leftrightarrow l_1 \vee l_2$ qui permet à une résolvante $l_1 \vee l_2 \vee l_3 \vee l_4$ de se réduire à $x \vee l_3 \vee l_4$. x est remplacé plus tard par $l_1 \vee l_2$ lorsque la clause qui le contient est binaire. Un exemple d'amincissements menant à une preuve étroite se trouve en figure **??**.

Théorème 1 ER3 *p-simule* ER.

Preuve 1 *La proposition 4 de [?] montre qu'il existe une fonction de réduction d'une preuve arborescente dans* ER *en une preuve arborescente dans* ER3 *dont la complexité est polynômiale. Comme* ER *est un exemple de système de Frege et qu'il est montré dans [?] que les preuves des systèmes de Frege sont réductibles en temps polynômial en des preuves arborescentes, on en conclut directement que* ER3 *p-simule* ER. □

On trouvera par exemple dans [?] une explication de la façon dont une preuve dans ER peut être transformée en preuve arborescente en temps polynômial.

Conséquences théoriques et pratiques

Chercher une réfutation dans Res, c'est chercher une séquence de résolutions qui aboutit à la clause vide. Une conséquence intéressante de notre résultat est qu'on peut maintenant envisager cette recherche de réfutation tout autrement : chercher une réfutation dans ER3, c'est chercher l'ensemble des extensions telles qu'une suite de résolutions étroites (en nombre cubique) aboutit à la clause vide. Ainsi, l'ensemble des extensions d'une réfutation constitue un certificat d'insatisfiabilité dont la complexité de vérification est une fonction cubique du nombre d'extensions et de clauses d'entrée.

Si P $\neq$ co-NP, certaines instances nécessitent un temps superpolynômial pour que leur réfutation soit découverte par ER3, ce qui implique que le nombre d'extensions nécessaires est aussi superpolynômial. Comme une extension peut être vue comme un lemme permettant de renommer des morceaux de preuve identiques afin de réduire la taille de la preuve, cela signifie qu'il existerait des instances dont les réfutations ne pourraient être raccourcies que par un nombre exponentiel de lemmes. Les lemmes n'y auraient donc qu'un effet marginal voire nul. La structure des preuves serait trop irrégulière pour être suffisamment "factorisable".

D'un point de vue pratique, on peut imaginer un algorithme de recherche de réfutation basé sur ER3 et n'autorisant qu'un nombre polynômial d'extensions. Il ne serait pas complet mais il ne serait incapable de décider que des instances de toute façon trop longue à traiter par ER3. Il chercherait donc une suite de longueur polynômiale d'extensions dont il vérifierait qu'elle complète la formule initiale de telle manière qu'une suite de résolutions étroites (en nombre cubique) permet d'inférer la clause vide. Cet algorithme pourrait être systématique (rechercher toutes les suites d'extensions possibles) ou procéder par réparation (proposer une suite d'extensions initiale et la modifier de proche en proche).

Cette approche de la recherche de réfutation diffère complètement des approches traditionnelles basées sur la résolution. La résolution étendue a toujours été considérée comme principalement de la résolution avec secondairement des extensions. Or, ER3 se voit naturellement comme des extensions complétées par de la résolution étroite. De ce fait, pour définir des algorithmes basés sur ER3, nous ne pouvons pas envisager de nous appuyer sur des algorithmes efficaces existants. C'est pourquoi il nous est apparu utile de chercher une variante de ER3 qui puisse à nouveau se voir comme de la "résolution avec des extensions".

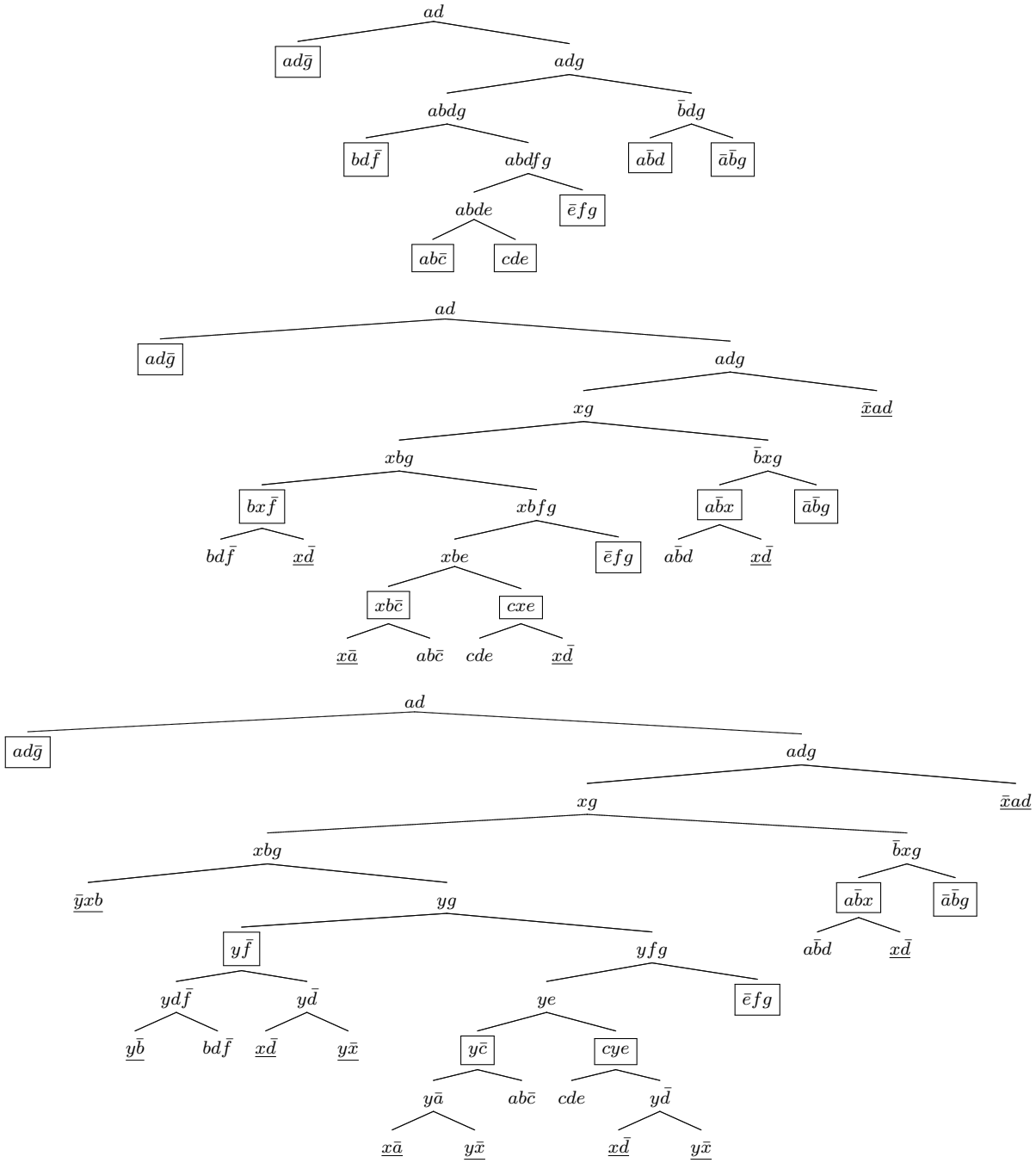


FIGURE 1 – Deux amincissements consécutifs menant à une preuve étroite. Les deux extensions sont $x \Leftrightarrow (a \vee d)$, raccourcissant $abdg$, et $y \Leftrightarrow (x \vee b)$, raccourcissant $xbfg$. Les clauses des extensions sont surlignées. Les clauses encadrées figurent les clauses d'entrée ou ce qu'elles sont devenues suite à l'intégration des extensions.

4 La Résolution à refragmentation

Voici une variante de ER3, que nous appellerons *Résolution à refragmentation*, dont la définition des règles permet une intégration facile de la résolution étendue dans n'importe quel solveur utilisant le mécanisme de la résolution. Son intérêt est de se présenter comme une autre façon de faire de la résolution. Ses deux règles sont :

- la règle de résolution étroite :

$$\frac{C_1 \qquad C_2}{C_1 \backslash \{l\} \cup C_2 \backslash \{\bar{l}\}}$$

  si $|C_1 \backslash \{l\} \cup C_2 \backslash \{\bar{l}\}| \leq 3$.
- la règle de refragmentation :

$$\frac{l \vee l_1 \vee l_2 \qquad \bar{l} \vee l_3 \vee l_4}{x \vee l_1 \vee l_3,\ \bar{x} \vee l_2 \vee l_4,\ x \vee \bar{l_2},\ x \vee \bar{l_4}}$$

  si l_1, l_2, l_3 et l_4 sont différents, où x est une nouvelle variable.

La règle de refragmentation permet en quelque sorte de scinder une résolvante $l_1 \vee l_2 \vee l_3 \vee l_4$ en deux clauses ternaires $x \vee l_1 \vee l_3$ et $\bar{x} \vee l_2 \vee l_4$, grâce aux clauses de l'extension $x \Leftrightarrow l_2 \vee l_4$. Elle correspond simplement à l'ajout des trois clauses de l'extension $\bar{x} \vee l_2 \vee l_4$, $x \vee \bar{l_2}$, $x \vee \bar{l_4}$ puis à une suite de trois résolutions étroites pour obtenir $x \vee l_1 \vee l_3$. C'est pourquoi nous pourrions aussi l'exprimer ainsi :

$$\frac{l \vee l_1 \vee l_2 \qquad \bar{l} \vee l_3 \vee l_4}{x \vee l_1 \vee l_3,\ x \Leftrightarrow l_2 \vee l_4}$$

Ce système de preuve n'est censé s'appliquer qu'à des instances 3-SAT. Il permet de ne générer que des clauses courtes.

Définition 1 *3-saturation d'une formule*
On appelle 3-saturation *d'une formule 3-SAT la complétion de cette formule avec toutes les résolvantes étroites possibles par application itérative de la règle de résolution étroite. Une formule est dite* 3-saturée *si la règle de résolution étroite ne peut pas (ou plus) s'appliquer.*

Comme nous l'avons vu, si n est le nombre de variables d'une formule, le nombre maximum de résolvantes est en $\Theta(n^3)$ donc la 3-saturation d'une formule 3-SAT se fait en temps polynômial.

Théorème 2 *La Résolution à refragmentation p-simule la Résolution étendue.*

Preuve 2 *Il suffit de montrer que la Résolution à refragmentation p-simule* ER3. *Or, comme les deux systèmes de preuve ne différent que par une seule règle, la règle de refragmentation remplaçant la règle d'extension, il suffit de démontrer qu'une suite polynômialement bornée d'applications de la règle de résolution étroite et de la règle de refragmentation permet d'obtenir n'importe quelle extension utile à la recherche d'une réfutation.*
Définissions le graphe de résolution (X, V) où l'ensemble des sommets X sont les clauses issus d'une instance (clauses d'entrée, résolvantes et clauses d'extension) et l'ensemble des arcs V est tel que $(C_1, C_2) \in V$ ssi on peut appliquer la règle de résolution entre C_1 et C_2. Effectuons (en temps polynômial) une 3-saturation de la formule. Dès lors, seule la règle de refragmentation peut s'appliquer. Considérons dans un premier temps le cas où le graphe de résolution est connexe. Pour chaque couple de littéraux (l_1, l_2) tels qu'ils ne portent pas sur la même variable, deux cas sont possibles : soit il existe une clause qui les contient tous les deux, soit ils n'apparaissent que séparément dans des clauses différentes. Dans le premier cas, nous considérons le cycle (qui est une chaîne revenant sur elle-même) le plus court partant et revenant à la clause. Dans le deuxième cas, considérons la chaîne la plus courte permettant de relier une clause ternaire contenant l_1 à une autre clause ternaire contenant l_2. Nous appelons distance entre l_1 et l_2 la longueur de cette chaîne (ou de ce cycle). Nous procédons par récurrence sur la longueur de la chaîne. Si la longueur est 1, les deux clauses se résolvent grâce à la règle de refragmentation et on obtient les clauses de l'extension $x \Leftrightarrow l_1 \vee l_2$. Si la longueur k est supérieur à 1, considérons les trois premiers éléments de la chaîne : $l_1 \vee a \vee b$, $\bar{b} \vee c \vee d$, $\bar{d} \vee e \vee f$. L'application de la règle de refragmentation aux deux premières clauses permet notamment d'obtenir $x \vee l_1 \vee d$. La chaîne commençant par $x \vee l_1 \vee d$ et $\bar{d} \vee e \vee f$ et suivie par les clauses restantes de la chaîne précédente est de longueur $k-1$ et permet de relier l_1 à l_2. Il suffit donc de répéter l'opération $k-1$ fois pour obtenir les clauses de $x \Leftrightarrow l_1 \vee l_2$. Vérifions maintenant que k est polynômial en n le nombre de variables de l'instance. Il faut prendre en compte le fait que des extensions ont pu compléter le graphe de résolution auparavant. La distance maximale d_1 entre deux littéraux de la formule initiale est inférieur au nombre de clauses qui est en $O(n^3)$. La distance d entre deux littéraux d'extension est inférieure à $d_1 + 2d_2$, où d_2 est la distance maximale entre un littéral d'extension et un littéral de la formule initiale. Il nous reste donc à montrer que d_2 est polynômial en n. Une extension $z \Leftrightarrow x \vee y$ ajoute trois clauses qui complètent le graphe de résolution. Etant donnée l'extension $z \Leftrightarrow x \vee y$, la distance $d(z, l)$ entre z et un littéral l de l'instance initiale est bornée par $1 + min(d(x, l), d(y, l))$. $d(z, l)$ est donc borné par le logarithme du nombre d'extensions (qui peut être une exponentielle de n) donc $d(z, l)$ est borné par un polynôme en n.
Dans le cas où le graphe de résolution n'est pas

connexe, chaque composante connexe représente une sous-formule indépendante. L'ajout d'une extension portant sur des littéraux appartenant à deux sous-formules indépendantes est inutile car la preuve la plus courte se fait en utilisant les clauses d'une seule des sous-formules. □

L'intérêt de la Résolution à refragmentation est qu'elle est facile à intégrer à un solveur. A la place d'une résolution standard, le solveur applique l'une des deux règles selon la taille attendue de la résolvante produite. Il faut cependant noter que l'application de la règle de refragmentation n'est pas déterministe une fois les deux clauses sélectionnées contrairement à la règle de résolution. En effet, à partir des clauses $l \vee l_1 \vee l_2$ et $\bar{l} \vee l_3 \vee l_4$, la règle de refragmentation peut s'appliquer de 4 façons différentes et produire au choix les clauses suivantes :

- $x \vee l_1 \vee l_3$, $\bar{x} \vee l_2 \vee l_4$, $x \vee \bar{l_2}$, $x \vee \bar{l_4}$ (extension $x \Leftrightarrow l_2 \vee l_4$)
- $x \vee l_1 \vee l_4$, $\bar{x} \vee l_2 \vee l_3$, $x \vee \bar{l_2}$, $x \vee \bar{l_3}$ (extension $x \Leftrightarrow l_2 \vee l_3$)
- $x \vee l_2 \vee l_3$, $\bar{x} \vee l_1 \vee l_4$, $x \vee \bar{l_1}$, $x \vee \bar{l_4}$ (extension $x \Leftrightarrow l_1 \vee l_4$)
- $x \vee l_2 \vee l_4$, $\bar{x} \vee l_1 \vee l_3$, $x \vee \bar{l_1}$, $x \vee \bar{l_3}$ (extension $x \Leftrightarrow l_1 \vee l_3$)

C'est à ce choix d'extension parmi 4 possibles à chaque résolution que peut se résumer l'augmentation de la combinatoire de la Résolution étendue par rapport à la Résolution standard. La Résolution à refragmentation peut par exemple facilement s'intégrer à un algorithme de type CDCL car chaque clause apprise peut s'interpréter en terme d'une suite de résolutions. Il suffit de remplacer chaque résolution par une résolution à refragmentation (à chaque fois que la résolvante serait longue) en choisissant une des 4 extensions selon une heuristique qu'il reste à trouver.

Il est à noter que la Résolution à refragmentation peut sembler ne pas véritablement différer d'une simple réécriture artificielle d'une clause quaternaire en deux clauses ternaires, à la manière dont on transforme toute instance SAT en instance 3-SAT. Ce serait vrai si la règle de refragmentation était

$$\frac{l \vee l_1 \vee l_2 \qquad \bar{l} \vee l_3 \vee l_4}{x \vee l_1 \vee l_3,\ \bar{x} \vee l_2 \vee l_4}$$

(sans les clauses $x \vee \bar{l_2}$ et $x \vee \bar{l_4}$). Il n'est pas difficile de montrer qu'avec cette règle de refragmentation "affaiblie", notre système ne serait équivalent qu'à la Résolution. Ce sont les deux clauses $x \vee \bar{l_2}$ et $x \vee \bar{l_4}$ de la règle de refragmentation qui donnent toute sa puissance à notre système : elles permettent d'obtenir d'autres clauses où x apparaît (l_2 et/ou l_4 ayant été remplacés par x).

5 Conclusion et perspectives

Nous avons montré que la Résolution étendue étroite (`ER3`), qui interdit la génération de résolvantes de longueur supérieure à 3, conserve toute la puissance de la Résolution étendue (`ER`). De plus, comme les instances de problèmes nécessitant un nombre superpolynômial d'extensions pour être résolus dans `ER3` seraient intraitables, il est raisonnable de vouloir borner polynômialement le nombre d'extensions. Il devient alors envisageable de définir des algorithmes basés uniquement sur la recherche des bonnes extensions, la suite des résolutions étroites à appliquer ensuite ne servant que de certificat polynômial d'insatisfiabilité. Ce type d'approche est nouveau car la combinatoire très élevée de `ER` ne permettait d'envisager jusqu'à présent qu'un nombre limité d'ajout d'extensions, ce qui bridait très fortement la puissance du système de preuve. Une perspective évidente de notre travail théorique est donc de définir des algorithmes (systématiques ou de réparation) efficaces basés sur cette approche. Mais comme cette approche impliquerait de définir des algorithmes entièrement nouveaux, nous avons aussi défini la Résolution à refragmentation, dans le but de permettre une intégration facile de la résolution étendue étroite dans n'importe quel algorithme utilisant la règle de résolution.

Nous avons donné les moyens théoriques de rendre l'utilisation de la Résolution étendue beaucoup plus efficace qu'auparavant. Mais il reste encore beaucoup à faire avant d'espérer obtenir un algorithme compétitif avec les meilleures techniques existantes. En effet, l'augmentation de la combinatoire de la Résolution étendue par rapport à la Résolution standard est dû au choix des extensions à faire, qui est une question difficile encore très peu traitée. La piste que nous allons suivre pour tenter de résoudre cette question est la suivante. Dans la mesure où nous devons trouver les extensions qui vont faire que la formule sera résoluble par une 3-saturation, il faut d'abord que nous puissions caractériser structurellement ce qu'est une instance résoluble par 3-saturation.

Références

[1] Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning sat solvers. In *AAAI*, 2010.

[2] Gilles Audemard and Laurent Simon. Gunsat : A greedy local search algorithm for unsatisfiability. In *IJCAI*, pages 2256–2261, 2007.

[3] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *J. ACM*, 48(2) :149–169, 2001.

[4] Vasek Chvátal and Endre Szemerédi. Many hard examples for resolution. *J. ACM*, 35(4) :759–768, 1988.

[5] Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8 :28–32, 1976.

[6] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3) :201–215, 1960.

[7] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39 :297–308, 1985.

[8] Jinbo Huang. Extended clause learning. *Artif. Intell.*, 174(15) :1277–1284, 2010.

[9] M. Jarvisalo. On the relative efficiency of dpll and obdds with axiom and join. In *CP 2011*, pages 429–437, 2011.

[10] Robert A. Kowalski and Donald Kuehner. Linear resolution with selection function. *Artif. Intell.*, 2(3/4) :227–260, 1971.

[11] J. Krajicek. Speed-up for propositional frege systems via generalizations of proofs. *Commentationes Mathematicae Universitas Carolinae*, 30(1) :137–140, 1989.

[12] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *IJCAI (1)*, pages 366–371, 1997.

[13] Nicolas Prcovic. Résolution étendue et largeur de réfutation de formules sat. In *Septièmes Journées Francophones de Programmation par Contraintes*, pages 271–280, 2011.

[14] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1) :23–41, 1965.

[15] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logics*, pages 115–125, 1968.

[16] Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1) :209–219, 1987.

Une stratégie de recherche basée sur la substituabilité

Mohamed Rezgui **Jean-Charles Régin** **Arnaud Malapert**

Laboratoire i3S
Université de Nice-Sophia Antipolis

rezgui@i3s.unice.fr {Jean-Charles.REGIN, arnaud.malapert}@unice.fr

Résumé

Nous introduisons une nouvelle stratégie de recherche pour énumérer toutes les solutions d'un problème de satisfaction de contraintes. L'idée principale de cette stratégie consiste à énumérer des solutions génériques à partir desquelles toutes les solutions peuvent être efficacement calculées. Les solutions génériques contiennent des valeurs qui sont substituables à toutes les autres. Notre stratégie provoque l'apparition des valeurs substituables. Ainsi, à la différence d'une stratégie de recherche classique, notre méthode économise du temps en générant seulement quelques solutions génériques. Nous montrons expérimentalement que notre approche donne des résultats intéressants sur des problèmes ayant un grand nombre de solutions.

1 Introduction

Les problèmes de satisfaction de contraintes (CSP) constituent un cadre formel simple pour représenter et résoudre des problèmes en intelligence artificielle et en recherche opérationnelle. Dans cet article, nous nous focalisons sur le calcul de toutes les solutions d'un problème. Calculer toutes les solutions d'une instance permet par exemple de représenter certains sous-problèmes par une contrainte de table contenant comme combinaisons l'ensemble des solutions de ces sous-problèmes. Nous proposons une nouvelle stratégie de recherche basée sur la notion de substituabilité des valeurs que l'on nomme Substituability Based Search (SBS). **L'idée est de détecter les valeurs substituables et de forcer leur apparition afin d'obtenir des solutions génériques qui seront ensuite énumérées pour générer toutes les solutions. On espère ainsi parcourir moins de noeuds de l'arbre de recherche**.

Nous pouvons illustrer cela sur l'exemple suivant. On considère un problème Π impliquant une variable x et a et b, deux valeurs appartenant au domaine de x que l'on note $D(x)$. Supposons que (x, a) soit compatible avec toutes les valeurs de toutes les contraintes impliquant x alors a est substituable à toutes les autres valeurs de $D(x)$. Cela signifie qu'il ne peut y avoir de solution impliquant (x, b) qui ne soit pas une solution avec (x, a). Par ailleurs, si une affectation impliquant (x, a) est une solution alors il est facile de tester si cette affectation reste une solution quand on remplace (x, a) par (x, b). En effet, il suffit de tester la validité des contraintes. Ainsi, en cherchant toutes les solutions impliquant seulement (x, a), on peut énumérer toutes les solutions impliquant les autres valeurs de $D(x)$.
La SBS est basée sur cette idée. De plus, SBS va provoquer cette substituabilité en s'inspirant du principe de l'algorithme de Bron & Kerbosh [1]. On procède de la manière suivante. Soit (y_k, b_k), les couples (variable, valeur) non-supports de (x, a). En instanciant d'abord les couples (y_k, b_k) avant (x, a), on va provoquer la substituabilité de (x, a). En effet, au moment où (x, a) sera choisi, les couples (y_k, b_k) auront disparus et (x, a) sera compatible avec toutes les valeurs restantes des $D(y_k)$. Ces valeurs sont mises dans un ensemble associé à x que l'on nomme $TSIE_x$. Une solution générique est constituée d'ensembles de TSIE. L'énumération d'une solution se fait de la manière suivante. Lorsque toutes les variables sont instanciées, il faudra énumérer de manière efficace les solutions en vérifiant la compatibilité de chaque valeur a contenue dans $TSIE_x$ avec les autres valeurs contenues dans les autres ensembles $TSIE_{x'}$. Les ensembles $TSIE_x$ qui contiennent des valeurs compatibles en leur sein et aux autres valeurs de $TSIE_{x'}$ forment des séquences $Global\ Cut\ Seed$ (GCS), notion reprise de l'article [2]. Ainsi, nous proposons dans cet article un algo-

rithme permettant de calculer ces GCS qui seront ensuite énumérées pour générer toutes les solutions. L'article est organisé comme suit. Tout d'abord, nous rappelons certaines notions. Ensuite, nous exposons les algorithmes utilisés par SBS. Puis, nous présentons les résultats expérimentaux relatant les avantages et les inconvénients de SBS. Enfin, nous concluons sur les apports et les différentes possibilités qu'offre SBS.

2 Notations

Dans les sections suivantes, nous utiliserons les notations suivantes. Soit $X = [x_1, x_2, ..., x_n]$, la liste des variables d'un CSP binaire Π et $D(x_i)$, le domaine associé à une variable x_i. x_i et x_j sont deux variables distinctes avec $i < j$. Une valeur a_i représente une valeur de $D(x_i)$. On note (x_i, a_i), un couple (variable, valeur) et $ListNoSupports(x_i, a_i) = \{(y_1, b_1), ..., (y_m, b_m)\}$, l'ensemble des couples ayant des valeurs non compatibles avec (x_i, a_i). Ces couples sont les conflits de (x_i, a_i). Une solution de Π est noté Sol_Π.

3 SBS : Search Based Substituality

3.1 Substituabilité

Definition 1 *[3] Soit a_i et b_i, deux valeurs de $D(x_i)$. On dit que a_i est* **substituable** *à b_i si et seulement si toute solution contenant b_i demeure une solution de Π si on remplace b_i par a_i.*

Cette notion a été introduite par Freuder [3] pour accélérer la recherche de solution pour les CSPs en liant deux ou plusieurs valeurs d'une même variable. Par exemple, on a deux couples distincts $(x_i, 1)$ *et* $(x_i, 2)$, la valeur 1 est substituable à la valeur 2 si et seulement si pour toute solution Sol_Π avec $x_i = 2$, il existe une solution Sol'_Π avec $x_i = 1$. SBS utilise ce principe pour compresser des valeurs dans des solutions génériques qui seront plus tard utilisées pour générer toutes les solutions. Pour cela, l'algorithme SBS va provoquer l'apparition des valeurs substituables. Dans les sections suivantes, nous allons détailler l'approche classique d'énumération de solutions ainsi que l'approche SBS.

3.2 Énumération classique de solutions

L'énumération des solutions sur un arbre binaire se fait de la façon suivante. On choisit une variable x_i, puis on choisit une valeur a_i ensuite on instancie $x_i = a_i$ et on propage les conséquences de cette instanciation. Lors de la présence d'un échec (une variable quelconque du CSP Π ayant un domaine vide), on fait un retour arrière (backtrack) et on lance la propagation avec $x_i \neq a_i$. Lorsqu'on prend une décision ($x_i = a_i$ ou $x_i \neq a_i$), on crée un point de choix. Ce dernier est également représenté par un noeud de l'arbre de recherche. Après chaque point de choix, on répète le processus. Lorsque toutes les variables sont instanciées, on a trouvé une solution. L'algorithme ClassicSearch (1) illustre cette approche. On utilise des stratégies de sélection de variables et de valeurs pour sélectionner le couple (variable, valeur) à chaque point de choix. Lorsque toutes les variables sont instanciées, on ajoute la solution trouvée puis on fait un retour arrière pour chercher une nouvelle solution.

Algorithm 1: function CLASSICSEARCH

```
CLASSICSEARCH(X)
REQUIRE : X ← list of variables
if nbAssignments = |X| then
    addSolution(X)
else
    x_i ← selectVariable(X)
    a_i ← selectValue(D(x_i))
    saveState()
    assign(x_i, a_i)
    if propagate(x_i = a_i) then
        ClassicSearch(X)
    unassign(x_i, a_i)
    restoreState()
    saveState()
    if propagate(x_i ≠ a_i) then
        ClassicSearch(X)
    restoreState()
```

3.3 Principe de l'algorithme SBS

Definition 2 *Un* **Tuple Sequence of Interchangeable Elements** *noté TSIE, est un n-tuples $\Delta = (\delta_1, ..., \delta_n)$, où chaque δ_i est un ensemble de valeurs non vide, tel que chaque n-tuple $(v_1, ..., v_n) \in \delta_1, ..., \delta_n$ et $v_n \in D_n$ est soit une solution Sol_Π ou soit dans un état impossible pour le problème Π.*

L'algorithme SBS (2) a une approche différente. Il n'énumère pas explicitement toutes les solutions. Il calcule des solutions génériques à partir desquelles l'ensemble des solutions sera calculé. Á un niveau donné, l'algorithme classique sélectionne une variable puis essaie successivement chaque valeur de cette variable. L'algorithme SBS procède différemment. On identifie le couple (x_i, a_i) qui a le moins de conflits. On instancie d'abord chaque conflit de (x_i, a_i) et on instancie (x_i, a_i). Á ce moment, Les autres valeurs de $D(x_i)$ n'ont pas besoin d'être instanciées avec x_i, car elles ne sont plus en conflit avec les valeurs des $D(x_j)$. En procédant ainsi, on espère réduire l'espace de recherche. Quand on instancie (x_i, a_i), on place les valeurs restantes de $D(x_i)$ dans $TSIE$. Enfin, on instancie (x_i, a_i). Si le nombre de conflits est strictement inférieur au

Algorithm 2: function SBS

```
SBS(X)
REQUIRE : X ← list of variables
if nbAssignments = |X| then
    TSIE-enumeration(TSIE)
else
    (x_i, a_i) ← getVariableValueMinConflicts()
    for each (y, b) ∈ ListNoSupports(x_i, a_i) do
        TSIE[y] ← TSIE[y] ∪ {b}
        saveState()
        assign(y, b)
        if propagate(y = b) then
            SBS(X)
        unassign(y, b)
        restoreState()
        TSIE[y] ← TSIE[y] − {b}
        if not propagate(y ≠ b) then
            return
    TSIE[x_i] ← TSIE[x_i] ∪ D(x_i)
    saveState()
    assign(x_i, a_i)
    if propagate(x_i = a_i) then
        SBS(X)
    unassign(x_i, a_i)
    restoreState()
    TSIE[x_i] ← TSIE[x_i] − D(x_i)
```

Algorithm 3: function TSIE-ENUMERATION

```
TSIE-ENUMERATION(TSIE) : generic solution
S_n ← TSIE[n]
for each i from n − 1 to 1 do
    S_i ← ∅
    for each GCS ∈ S_{i+1} do
        newS ← ∅
        for each a_i ∈ TSIE[i] do
            // Calcul de la nouvelle GCS pour la valeur a_i
            newGCS[i] ← {a_i}
            for each j de i + 1 à n do
                newGCS[j] ←
                compatible(x_i, a_i, x_j) ∩ GCS[j]
            newS ← newS ∪ {newGCS}
        // Concaténation des GCS équivalents
        MERGEEQUIVALENTGCS(newS)
        S_i ← S_i ∪ newS
return S_1
```

nombre de valeurs dans le domaine alors SBS fera moins de choix qu'une méthode classique pour déterminer les solutions courantes contenant les éléments de $D(x_i)$. Une fois que toutes les variables sont instanciées, $TSIE$ devient une solution générique qu'il faudra énumérer en vérifiant la compatibilité des valeurs. Nous allons détailler cette énumération dans la section suivante.

4 Énumération des GCS

Definition 3 *[2] Une* **Global Cut Seed** *notée GCS, est un n-tuples $\Delta = (\delta_1, ..., \delta_n)$, où chaque δ_i est un ensemble de valeurs non vide, tel que chaque n-tuple $(v_1, ..., v_n), v_n \in \delta_1, ..., \delta_n$ et $v_n \in D_n$ est une solution du problème* Π. [1]

D'après la définition, on notera qu'il est facile d'énumérer les solutions dans des GCS. Pour énumérer les solutions, nous construisons des $Global\ Cut\ Seed$ (GCS) définies par Foccaci et Milano [2]. Chaque $TSIE_i$ contient une valeur Sub_i qui est est compatible avec toutes les valeurs des autres $TSIE_j$ avec $j > i$. De plus, cette valeur correspond à la valeur que l'on a explicitement instanciée. On peut donc facilement calculer toutes les solutions contenant cette valeur Sub_i. Pour les autres valeurs, c'est plus délicat, car elles peuvent être incompatibles avec des valeurs des autres $TSIE_j$. Il faut donc tester les compatibilités entre valeurs (3). Nous détaillons le déroulement de cet algorithme avec l'exemple suivant. On fixe X avec 5 variables et $TSIE = [\{a_1, b_1\}, \{a_2, b_2, c_2\}, \{c_3\}, \{d_4\}, \{e_5, f_5, g_5\}]$, la liste des ensembles de valeurs $TSIE_i$ générée par l'algorithme (2). Les contraintes de Π définissent les incompatibilités suivantes : e_5 incompatible avec b_2 et c_2, g_5 incompatible avec b_1 et c_3 incompatible avec b_1. Soit $GCS[i]$, l'ensemble des valeurs de la variable x_i de la GCS. L'algorithme satisfait la propriété suivante : à chaque fin d'étape i, on crée une liste S_i qui possède les ensembles des valeurs compatibles de x_i à x_n. Cette liste S_i regroupe des $Global\ Cut\ Seed$. En effet, à l'initialisation $i = 0$, on crée une $Global\ Cut\ Seed$ avec les valeurs supports de la dernière variable instanciée. Ensuite, à chaque étape i, on crée la liste S_i des $Global\ Cut\ Seed$ à partir des $Global\ Cut\ Seed$ générées à l'étape $i - 1$. Pour cela, on ajoute une valeur compatible si et seulement si après élimination des valeurs non compatibles aucun ensemble n'est vide. Pour calculer toutes les solutions de TSIE, on commence par la dernière étape qui génère la première GCS $[\{e_5, f_5, g_5\}]$, ensuite à chaque étape $i - 1$, on vérifie la compatibilité de chaque valeur de l'ensemble $TSIE_i$ avec les autres valeurs des autres ensembles précédents. Ainsi, à l'étape 4, on ajoute l'ensemble $\{d_4\}$, ce qui nous donne S_4 contenant $[\{d_4\}, \{e_5, f_5, g_5\}]$. On continue de la même façon pour la liste S_3, on a donc $[\{c_3\}, \{d_4\}, \{e_5, f_5, g_5\}]$. Á l'étape 2, on a e_5 qui est non compatible avec b_2 et c_2, ainsi la liste S_2 3 GCS : $[\{a_2\}, \{c_3\}, \{d_4\}, \{e_5, f_5, g_5\}]$, $[\{b_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$, $[\{c_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$.

1. Nous avons considéré une partie de la définition décrite par [2], car nous voulons réduire la notion de GCS seulement à une liste de tuples n'ayant que des valeurs compatibles

Enfin, à l'étape 1, on a g_5 qui est non compatible avec b_1 et c_3 qui est non compatible avec b_1. La liste S_1 contient :
$[\{a_1\},\{a_2\},\{c_3\},\{d_4\},\{e_5,f_5,g_5\}]$,
$[\{a_1\},\{b_2\},\{c_3\},\{d_4\},\{f_5,g_5\}]$,
$[\{a_1\},\{c_2\},\{c_3\},\{d_4\},\{f_5,g_5\}]$,
$[\{b_1\},\{b_2\},\{c_3\},\{d_4\},\{f_5,g_5\}]$,
$[\{b_1\},\{c_2\},\{c_3\},\{d_4\},\{f_5\}]$.
La GCS $[\{b_1\},\{a_2\},\{\},\{d_4\},\{e_5,f_5,g_5\}]$ n'est pas valide, car il y a la présence d'un ensemble vide. Après la génération de cinq GCS à l'étape 1, on les énumère afin de générer toutes les solutions émanant de cette solution générique. Ainsi, on aura au final dix solutions :
$[\{a_1\},\{a_2\},\{c_3\},\{d_4\},\{e_5\}]$,
$[\{a_1\},\{a_2\},\{c_3\},\{d_4\},\{f_5\}]$,
$[\{a_1\},\{a_2\},\{c_3\},\{d_4\},\{g_5\}]$,
$[\{a_1\},\{b_2\},\{c_3\},\{d_4\},\{f_5\}]$,
$[\{a_1\},\{b_2\},\{c_3\},\{d_4\},\{g_5\}]$,
$[\{a_1\},\{c_2\},\{c_3\},\{d_4\},\{f_5\}]$,
$[\{a_1\},\{c_2\},\{c_3\},\{d_4\},\{g_5\}]$,
$[\{b_1\},\{b_2\},\{c_3\},\{d_4\},\{f_5\}]$,
$[\{b_1\},\{b_2\},\{c_3\},\{d_4\},\{g_5\}]$,
$[\{b_1\},\{c_2\},\{c_3\},\{d_4\},\{f_5\}]$.
Dans cet exemple, la compression des solutions est d'un facteur 2, car on génère 5 GCS pour 10 solutions.
On remarque que la génération de nouvelles GCS peut amener à avoir des listes de GCS identiques. En effet dans l'exemple présenté, on peut observer que pour la liste S_2, on a deux GCS $[\{b_2\},\{c_3\},\{d_4\},\{f_5,g_5\}]$, $[\{c_2\},\{c_3\},\{d_4\},\{f_5,g_5\}]$ qui ont en commun $[\{c_3\},\{d_4\},\{f_5,g_5\}]$. Pour éviter la redondance des GCS, l'algorithme appelle la fonction MERGEEQUIVALENTGCS($newS$) qui recherche les GCS identiques pour la nouvelle GCS créée. Pour cela, on transforme la GCS en une séquence de valeurs. Ensuite, on fait un tri lexicographique pour déterminer s'il y a des égalités entre les GCS et on concatène les GCS identiques. La complexité de notre algorithme est la suivante. Le coût du tri lexicographique est $O(n\log(n))$ comparaisons. Chacune coûtant la somme des valeurs de la GCS. Dans notre cas, on a d éléments considérés et la taille de la chaine maximale est nd. On aura donc un coûte en $O(d\log(d)nd)$, soit $O(nd^2\log(d))$. Dans notre exemple, on aura donc après concaténation des 2 GCS $[\{b_2\},\{c_3\},\{d_4\},\{f_5,g_5\}]$, $[\{c_2\},\{c_3\},\{d_4\},\{f_5,g_5\}]$, une seule GCS $[\{b_2,c_2\},\{c_3\},\{d_4\},\{f_5,g_5\}]$. On peut essayer de concaténer les GCS de plusieurs façons différentes, mais il faut obligatoirement avoir des solutions uniques sinon l'énumération peut devenir difficile.

5 Résultats expérimentaux

Nous présentons dans cette section une évaluation expérimentale de nos algorithmes. Nous proposons de comparer l'approche SBS et l'approche classique d'énumération de solutions avec des différentes stratégies de sélection de couples (variable, valeur). Ensuite, nous évaluons l'efficacité de la concaténation des GCS. Toutes les expériences sont effectuées sur un solveur ad-hoc avec une machine disposant d'un processeur Intel Core i7 de quatre coeurs cadencés à 2,2 GHz. Chaque expérience est exécutée sur un seul coeur. On note $MinDom$, la stratégie de sélection de variables qui sélectionne la variable x_i ayant la plus petite taille du domaine. $MinConflict$ et $MaxConflict$ sont deux stratégies de sélection de valeurs qui sélectionnent la valeur a_i de $D(x_i)$ ayant respectivement le minimum et le maximum de conflits. Nous avons testé SBS et les différentes stratégies de recherche $MinDom/MinConflict$ et $MinDom/MaxConflict$. D'après les résultats, nous n'avons pas observé de différences significatives dans les temps de résolution entre $MinDom/MinConflict$ et $MinDom/MaxConflict$ sur l'ensemble des instances résolues. Ainsi, nous considérons seulement la comparaison entre SBS et $MinDom/MaxConflict$. La génération des instances se fait de manière aléatoire en fixant un nombre de variables V, un nombre de valeurs par domaine D, un nombre de contraintes en extension C, et une dureté des contraintes T ($tightness$ en anglais) qui définit le nombre de tuples non autorisés pour chaque contrainte C. Nous considérons seulement des instances ayant plus de 1000 solutions. On rappelle que les problèmes faiblement contraints avec une $tightness$ faible présentent beaucoup de solutions alors que ceux fortement contraints donc avec une $tightness$ élevée ont peu de solutions. Dans

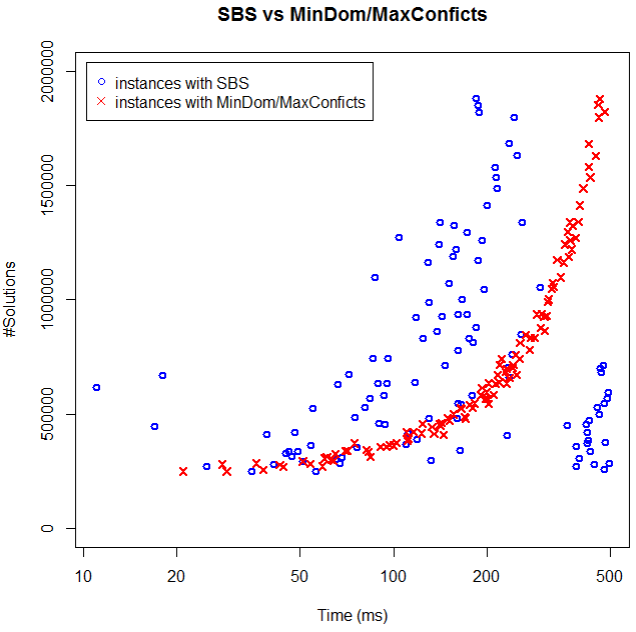


FIGURE 1 – Comparaison des temps de résolution entre SBS et $MinDom/MaxConflicts$

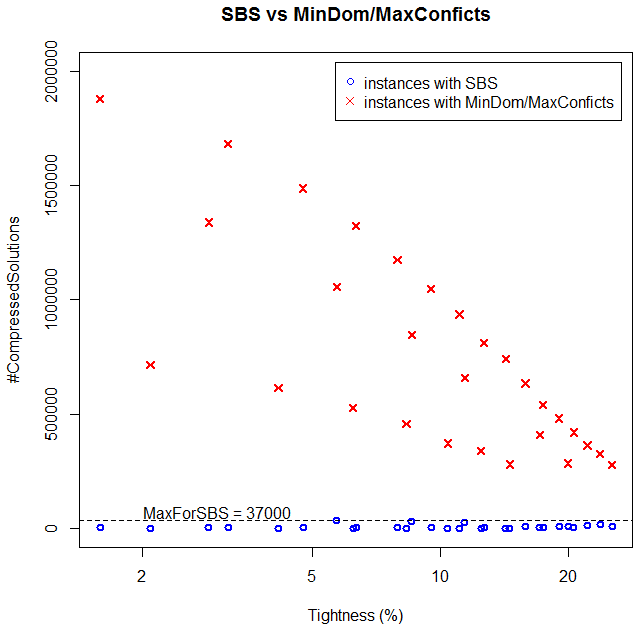


FIGURE 2 – Comparaison des solutions compressées entre SBS et $MinDom/MaxConflicts$

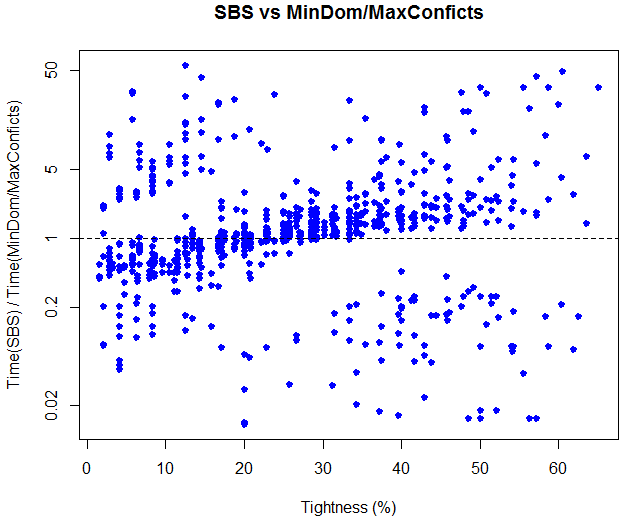


FIGURE 3 – Comparaison des ratios entre SBS et $MinDom/MaxConflicts$

les figures suivantes, chaque instance résolue est représentée par un point. Nous avons vérifié que le nombre de solutions trouvées pour chaque instance est exactement le même entre SBS et $MinDom/MaxConflict$.
La figure 1 compare les temps des deux méthodes pour énumérer toutes les solutions de chaque instance. Ici, SBS utilise la concaténation des GCS dans l'énu-

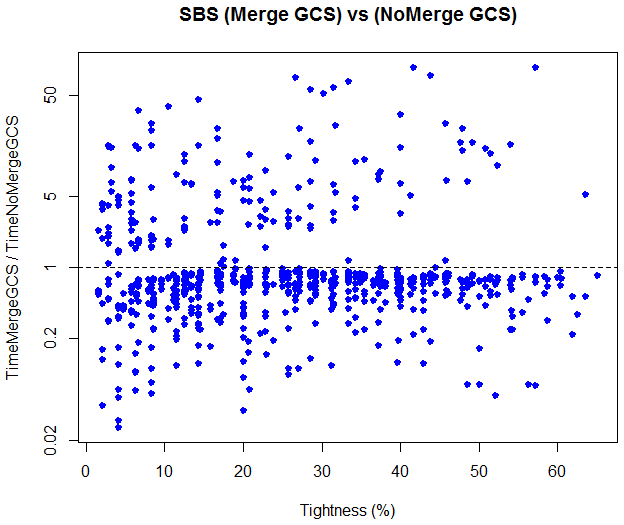


FIGURE 4 – Comparaison des ratios entre SBS avec et sans concaténation des GCS

mération des TSIE. La courbe décrite par la stratégie $MinDom/MaxConflict$ est homogène ainsi plus l'instance possède de solutions et plus le temps est long. Quant aux instances résolues par SBS, elles ont des temps de résolutions variables et sont éparpillées entre le dessous et le dessus de la courbe de $MinDom/MaxConflict$. Ainsi, SBS peut résoudre plus rapidement des instances que $MinDom/MaxConflict$ et plus lentement pour d'autres. Nous allons voir en détail ce qui se passe pour les instances résolues entre SBS et $MinDom/MaxConflict$. La figure 2 compare le nombre de solutions compressées par SBS (solutions génériques) et le nombre de solutions que trouve généralement une stratégie de recherche classique comme $MinDom/MaxConflict$ en fonction de la $tightness$ ici calculée en pourcentage. La compression des solutions avec SBS est impressionnante lorsque la $tightness$ est faible (pour une $tightness$ évaluée inférieure à 25%). En effet, le facteur de compression varie entre 2 et 50 (rapport entre nombre de solutions et le nombre de solutions compressées). On constate qu'au fur et à mesure que celle-ci augmente, la compression de solutions commence à perdre en efficacité. Le facteur de compression est quasiment nul à partir de 50% de $tightness$. Ainsi, déterminer la $tightness$ du problème est très importante afin de pouvoir évaluer à l'avance la stratégie de recherche la plus efficace pour calculer toutes les solutions. Nous avons testé l'efficacité de la concaténation des GCS évoquée dans l'algorithme (3). La figure 3 compare les ratios entre le temps de résolution de SBS et celui de $MinDom/MaxConflict$ en fonction de la $tightness$ ici calculée en pourcentage. SBS utilise la concaténation des GCS dans l'énumération des TSIE. On trace la droite $y = 1$

pour évaluer les ratios sur le graphique. Si une instance est résolue plus rapidement avec SBS, il doit se trouver en dessous de la droite $y = 1$ et inversement. On remarque que pour beaucoup de problèmes faiblement contraints, SBS est beaucoup plus rapide que $MinDom/MaxConflict$ jusqu'à environ 25% de $tightness$. Après ce plafond, $MinDom/MaxConflict$ est plus efficace. C'est prévisible, car comme le montre la figure 2, ce plafond coordonne avec l'écart qui se réduit entre le nombre de solutions compressées par SBS et le nombre de solutions non compressées que trouve $MinDom/MaxConflict$.
Dans la figure 4, nous comparons les ratios entre le temps de résolution de SBS avec concaténation des GCS et celui de SBS sans concaténation des GCS en fonction de la $tightness$ ici calculée en pourcentage. On trace la droite $y = 1$ pour évaluer les ratios sur le graphique, ainsi si une instance est résolue plus rapidement avec SBS avec concaténation des GCS, il doit se trouver en dessous de la droite $y = 1$ et inversement. Avec une $tightness$ faible de l'ordre de moins de 5%, les deux approches sont équivalentes, mais dépassé ce plafond, on observe que pour la plupart des instances, la concaténation des GCS lors de leur énumération permet d'améliorer considérablement le temps de résolution (2 à 5 fois plus rapide).
L'ensemble de ces résultats expérimentaux montre que SBS est très efficace dans le calcul de toutes les solutions des problèmes peu contraints par rapport à une stratégie de recherche classique. Avec des $tightness$ allant de 0% à 25%, on réduit le temps de résolution par 3 en moyenne. La concaténation des GCS améliore sensiblement l'énumération des TSIE avec des gains de temps considérables (2 à 5 fois plus rapide pour la plupart des instances).

6 Conclusion

Nous avons proposé une nouvelle stratégie de recherche basée sur la substituabilité (SBS) pour calculer toutes les solutions d'une instance. D'après les résultats expérimentaux, SBS améliore les problèmes d'un facteur 2 à 5 en temps de résolution et d'un facteur de 2 à 50 en terme de mémoire. Mais, cette approche perd en efficacité quand la $tightness$ augmente au-delà de 25%. Nous avons comparé les différentes stratégies de recherche et nous avons évalué l'efficacité de la concaténation des GCS pour l'énumération des solutions. L'algorithme présenté ne considère que les contraintes binaires en extension, il serait intéressant de généraliser cette idée pour les CSP classiques invoquant notamment des contraintes globales.

Références

[1] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communications of the ACM*, pages 575–577, 1973.

[2] Focacci and Milano. Global cut framework for removing symmetries. pages 77–92, 2001.

[3] Eugene C. Freuder. Eliminating interchangeable values in csp. pages 227–233, 1991.

Compilation des QCSP

Igor Stéphan

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045, Angers, Cedex 01, France
igor.stephan@info.univ-angers.fr

Résumé

Nous proposons dans cet article un cadre formel pour la compilation des problèmes de satisfaction de contraintes quantifiées (QCSP). L'objectif d'une telle compilation est de répondre au problème du choix du prochain mouvement de manière polynomiale en temps même lorsque la solution courante n'est plus accessible. Nous établissons la sémantique de ce formalisme en terme d'interprétation en un QCSP. Nous en étudions les propriétés en particulier vis-à-vis du QCSP compilé. Nous spécifions deux algorithmes de compilation basés sur un algorithme de recherche. Le premier est imbriqué dans l'algorithme de recherche et reprend la structure inductive de la sémantique des QCSP ; le second est un analyseur de trace d'exécution d'un solveur QCSP.

Abstract

We propose in this article a framework for compilation of quantified constraint satisfaction problems (QCSP). The aim of this compilation is to answer the next move choice problem in polytime even when the current solution is no more accessible. We establish the semantics of this formalism by an interpretation to a QCSP. We specify two algorithms based on search algorithm. The first one is embedded into the search algorithm and is based on the inductive semantics of the QCSP. The second one is a QCSP solver trace analyser.

1 Introduction

Un problème de satisfaction de contraintes (ou CSP pour *Constraint Satisfaction Problem*) [13] est le problème de trouver une affectation de valeurs pour des variables qui satisfont un ensemble de contraintes. Un problème de satisfaction de contraintes *quantifiés* (ou QCSP pour *Quantified Constraint Satisfaction Problem*) [7] est une extension du problème de satisfaction de contraintes dans laquelle certaines variables sont quantifiées universellement (les autres variables étant quantifiées existentiellement) ; dans ce cadre, chaque variable prend sa valeur dans un domaine fini discret. Les variables universellement quantifiées peuvent être considérées comme étant une forme d'incertitude : les caprices de la nature ou les choix d'un adversaire dans un jeu à deux joueurs. Dans la seconde interprétation, calculer une solution à un QCSP c'est offrir au joueur existentiel (i.e. le joueur dont les coups sont représentés par des variables existentiellement quantifiées) une stratégie qui gagne à tous les coups. Tandis que résoudre un CSP est en général NP-complet, résoudre un QCSP est PSPACE-complet. La plupart des procédures de décision récentes pour les QCSP [1, 4, 11, 5] sont basées sur un algorithme de recherche [1]. Un tel algorithme choisit une variable, teste en affectant à la variable les différentes valeurs du domaine si les QCSP admette une solution et combine les résultats selon la sémantique du quantificateur associé à la variable.

En général, une base de connaissances est compilée une bonne fois pour toute dans un langage cible puis utilisée à la volée pour répondre à des questions ; l'objectif étant d'avoir une complexité bien moindre pour l'interrogation de la base de connaissances compilée que celle non compilée. L'objectif de notre compilation est de répondre au problème du choix du prochain mouvement de manière polynomiale en temps même lorsque la solution courante n'est plus accessible. Notre but est donc bien de compiler le QCSP et non simplement une solution. De part la complexité du calcul d'une simple solution, la recherche d'une forme compilée d'un QCSP est capitale. À notre connaissance, ce problème n'a pas été abordé dans le cadre des QCSP mais seulement dans le cadre de domaines connexes : la compilation des formules booléennes quantifiées (ou QBF pour *Quantified Boolean Formulas*) [16, 15, 10], la compilation des bases de connaissances [6, 9].

Ce présent article est une reformulation dans ces principes de [16, 15] mais est techniquement complè-

1. hormis [17] qui est basé sur une approche *bottom-up* et [12] qui est basé sur une traduction vers les formules booléennes quantifiées.

tement différent et novateur. Cet article est organisé ainsi : la section 2 pose l'ensemble des définitions nécessaires ; la section 3 définit notre proposition de cadre formel pour la compilation des QCSP ; la section 4 spécifie deux manières différentes et complémentaires de construire des bases pour un QCSP ; la section 5 dresse une conclusion et quelques perspectives.

2 Préliminaires

Le symbole $\exists$ représente le quantificateur existentiel et le symbole $\forall$ représente le quantificateur universel. Le symbole $\wedge$ représente la conjonction, le symbole $\top$ représente ce qui est toujours vrai et le symbole $\bot$ représente ce qui est toujours faux. Un QCSP est un n-uplet $(\mathbf{V}, \text{ordre}, \text{quant}, \mathbf{D}, \mathcal{C})$ dans lequel $\mathbf{V}$ est un ensemble de n variables, ordre est une bijection des variables sur $[1..n]$, quant est fonction de $\mathbf{V}$ dans $\{\exists, \forall\}$ (quant(v) est le quantificateur associé à la variable v), $\mathbf{D}$ est une fonction de l'ensemble de variables dans un ensemble de domaines $\{D(v_1), \ldots, D(v_n)\}$ où, pour chaque variable $v_i \in \mathbf{V}$, $D(v_i)$ est le domaine fini de l'ensemble des valeurs possibles ($D(v)$ est le domaine associé à la variable v), $\mathcal{C}$ est un ensemble de contraintes. Si $v_{j_1}, \ldots, v_{j_m}$ sont des variables d'une contrainte $c_j \in \mathcal{C}$ alors la relation associée à c_j est un sous-ensemble du produit cartésien $D(v_{j_1}) \times \ldots \times D(v_{j_m})$. Par la suite, nous noterons pour tout $i \in [1..n]$, $q_i = \text{quant}(v_i)$ et $D_i = D(v_i)$. Un QCSP $(\mathbf{V}, \text{ordre}, \text{quant}, \mathbf{D}, \mathcal{C})$ sur n variables sera décrit de façon plus compacte ainsi :

$$q_1 v_1 \ldots q_n v_n \bigwedge_{c_j \in \mathcal{C}} c_j$$

avec $v_1 \in D_1$, …, $v_n \in D_n$, ordre$(v_i) = i$, pour tout $i \in [1..n]$; $q_1 v_1 \ldots q_n v_n$ forme le lieur ; un lieur vide est noté ε.

Le QCSP $(\{x, y, z, t\}, \text{ordre}, \text{quant}, \mathbf{D}, \mathcal{C})$ avec

$$\begin{cases} \text{ordre} = \{(1, x), (2, y), (3, z), (4, t)\}, \\ \text{quant} = \{(x, \exists), (y, \exists), (z, \forall), (t, \exists)\}, \\ \mathbf{D} = \{(x, D(x)), (y, D(y)), (z, D(z)), (t, D(t))\}, \\ D(x) = D(y) = D(z) = D(t) = \{0, 1, 2\}, \\ \mathcal{C} = \{(x = (y * z) + t)\} \end{cases}$$

est, par exemple, noté : $\exists x \exists y \forall z \exists t (x = (y * z) + t)$ avec $x, y, z, t \in \{0, 1, 2\}$.

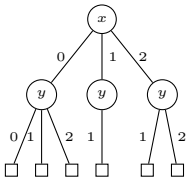
Dans la partie algorithmique (cf section 4), une variable v associée à un domaine D est notée v_D.

Dans un lieur, une séquence maximale homogène de quantificateurs forme un bloc ; le premier (et aussi le plus externe) est le plus à gauche.

L'ensemble $\mathcal{A}_i(Q)$ avec $v_1 \in D_1$, …, $v_n \in D_n$, $Q = q_1 v_1 \ldots q_n v_n$ pour $1 \leq i \leq n$ est l'ensemble des arbres dont

- toutes les feuilles sont étiquetées par le symbole $\square$ et à la profondeur i,
- tout nœud interne de profondeur k, $0 \leq k < i$ est étiqueté par la variable v_{k+1},
- tout arc reliant un nœud de profondeur k à un de ses nœuds fils est étiqueté par un élément de D_{k+1},
- tous les arcs reliant un nœud de profondeur k aux nœuds fils sont étiquetés différemment.

L'arbre suivant est, par exemple, un élément de l'ensemble $\mathcal{A}_2(\exists x \exists y \forall z \exists t)$ avec $x, y, z, t \in \{0, 1, 2\}$:



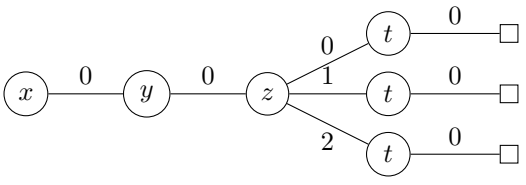
Soit un QCSP $(\mathbf{V}, \text{ordre}, \text{quant}, \mathbf{D}, \mathcal{C})$ tel que $\mathbf{V} = \{v_1, \ldots, v_n\}$, avec $v_1 \in D_1$, …, $v_n \in D_n$, alors un scénario est la séquence des étiquettes $val_1, \ldots, val_n$ sur une branche $(v_1, val_1), \ldots, (v_n, val_n)$, $val_i \in D_i$ pour tout i, $1 \leq i \leq n$, d'un arbre de $\mathcal{A}_n(q_1 v_1 \ldots q_n v_n)$ et une stratégie est un arbre de $\mathcal{A}_n(q_1 v_1 \ldots q_n v_n)$ tel que

- pour tous les nœuds étiquetés par une variable dont le quantificateur associé est existentiel admettent un unique nœud fils et
- pour tous les nœuds étiquetés par une variable dont le quantificateur associé est universel et le domaine associé est de taille k, admettent k nœuds fils.

Un scénario $val_1, \ldots, val_n$ pour un QCSP $(\mathbf{V}, \text{ordre}, \text{quant}, \mathbf{D}, \mathcal{C})$ tel que $\mathbf{V} = \{v_1, \ldots, v_n\}$ est un scénario gagnant si $(\bigwedge_{1 \leq i \leq n} v_i = val_i) \wedge (\bigwedge_{c_j \in \mathcal{C}} c_j)$ est vrai ; un tel scénario correspond à l'instantiation complète $[v_1 \leftarrow val_1]$, …, $[v_n \leftarrow val_n]$ qui est gagnante si l'instantiation satisfait toutes les contraintes. Une stratégie est gagnante si tous ses scénarios sont gagnants. En l'absence de quantificateur, la stratégie $\square$ est toujours une stratégie gagnante.

La scénario $0, 0, 2, 0$, qui correspond à l'instantiation complète $[x \leftarrow 0]$, $[y \leftarrow 0]$, $[z \leftarrow 2]$ et $[t \leftarrow 0]$, est un scénario gagnant, car $0 = (0 * 2) + 0$, ainsi que la stratégie suivante pour le QCSP

$$\exists x \exists y \forall z \exists t (x = (y * z) + t), x, y, z, t \in \{0, 1, 2\}$$



car $0 = (0 * 0) + 0$, $0 = (0 * 1) + 0$ et $0 = (0 * 2) + 0$.

Un QCSP $\forall x QC$ avec $x \in D$ admet une stratégie gagnante si et seulement si, pour tout $val \in D$, $Q(C \wedge (x = val))$ admet une stratégie gagnante et un QCSP $\exists x QC$ avec $x \in D$ admet une stratégie gagnante si et seulement si, pour au moins un $val \in D$, $Q(C \wedge (x = val))$ admet une stratégie gagnante.

3 Base pour QCSP

Une manière naïve de compiler un QCSP serait de stocker l'ensemble des stratégies gagnantes dans un ensemble mais l'approche est, du point de vue de la complexité spatiale, fortement exponentielle [2] et donc inappropriée ; par exemple, pour le QCSP $\forall x \forall y \exists z \exists t (x = (y * z) + t), x, y, z, t \in \{0, 1, 2\}$ il existe 324 stratégies gagnantes [3] ! Une autre manière de faire est de stocker un arbre contenant uniquement les scénarios présents dans les stratégies gagnantes ; cette approche est pauvre du point de vue de la représentation des connaissances puisqu'il n'y a pas d'accès direct aux possibilités d'une variable existentiellement quantifiée, hormis celles du premier bloc.

Nous définissons dans cette section notre formalisme pour représenter le résultat de la compilation d'un QCSP : la « base » ainsi que sa sémantique en terme de QCSP.

3.1 Définitions

Une base est intuitivement l'ensemble des stratégies organisées selon un mécanisme de garde pour chaque variable existentiellement quantifiée et chaque valeur du domaine de cette variable ; une telle garde est un arbre qui est l'expression des coups joués par les deux adversaires.

Définition 1 (base) *Une base est soit*
- *le symbole top_base*
- *le symbole bottom_base*
- *une paire $\langle Q \mid G \rangle$ avec $n > 0$, $Q = q_1 v_1 \dots q_n v_n$ et $G = [G_{e_1}, \dots, G_{e_m}]$ une liste telle que*
  - *$e_1, \dots, e_m$ est l'ensemble des indices des variables quantifiées existentiellement [4] ;*
  - *chaque G_{e_k}, $1 \leq k \leq m$ est une fonction dont le graphe $\{(val_1, a_1), \dots, (val_{j_k}, a_{j_k})\}$ est non vide, $val_1, \dots, val_{j_k} \in D(v_{e_k})$ et $a_1, \dots, a_{j_k} \in \mathcal{A}_{e_k}(Q)$.*

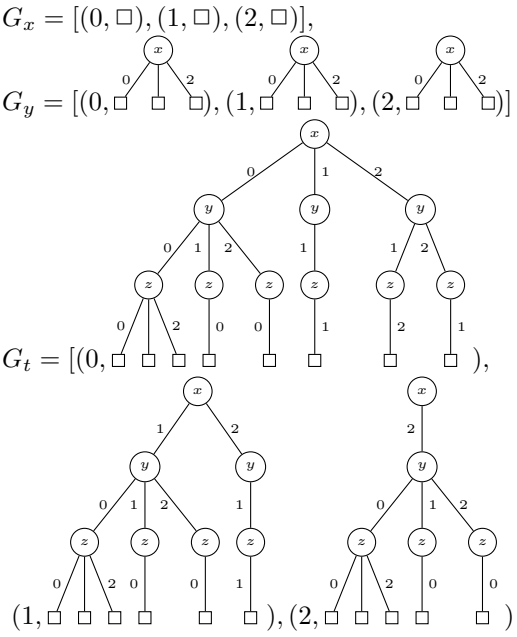


FIGURE 1 – Ensembles de gardes.

Dans ce qui suit, les bases *top_base* et *bottom_base* sont interprétées sémantiquement comme respectivement ce qui est toujours vrai et ce qui est toujours faux et algorithmiquement comme respectivement ce qui admet tout pour stratégie gagnante et ce qui n'admet aucune stratégie gagnante.

Exemple 1 *La figure 1 présente les ensembles de gardes G_x, G_y et G_z de la base $B = \langle \exists x \exists y \forall z \exists t \mid [G_x, G_y, G_t] \rangle$ avec $x, y, z, t \in \{0, 1, 2\}$.*

Nous avons choisi d'explorer uniquement les gardes sous la forme d'arbre mais il fait tout à fait possible, au prix d'une algorithmique plus complexe et d'un propos qui ne correspondait pas à ce qui voulait être présenté dans cet article, de les remplacer par, par exemple, des ADD (pour *Algebraic Decision Diagram* [2]) pour compiler aussi les gardes.

3.2 Interprétation

La sémantique d'une base s'exprime via une interprétation vers les QCSP. Nous interprétons tout d'abord les arbres présents dans les bases comme des contraintes sous formes de tables (i.e. des n-uplets de valeurs).

2. Dans les hypothèses de complexité classiques, la taille d'une quelconque représentation d'une stratégie gagnante est dans le pire des cas exponentielle par rapport au nombre de variables du QCSP [8]

3. Si n est la taille du domaine des variables, la fonction $\#(.)$ qui associe à un lieur Q son nombre de stratégies possibles est définie par $\#(\varepsilon) = 1$, $\#(\exists x Q) = n \times \#(Q)$ et $\#(\forall x Q) = \#(Q)^n$; donc pour le lieur $\forall x \forall y \exists z \exists t$ et des domaines pour les variables x, y, z et t de taille 3, il y $\#(\forall x \forall y \exists z \exists t) = ((3 \times 3 \times 1)^3)^3 = 387420489$ stratégies possibles.

4. i.e. $u_1, \dots, u_p$ est l'ensemble des indices des variables quantifiées universellement, $\{e_1, \dots, e_m\} \cup \{u_1, \dots, u_p\} = [1..n]$, $\{e_1, \dots, e_m\} \cap \{u_1, \dots, u_p\} = \emptyset$, $\text{quant}(v_{e_i}) = \exists$, pour tout i, $1 \leq i \leq n$, $\text{quant}(v_{u_i}) = \forall$, pour tout i, $1 \leq i \leq p$

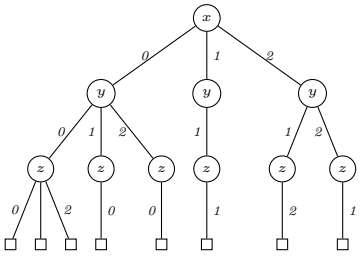
Définition 2 (interprétation d'un arbre) *L'interprétation d'un arbre a selon une valeur val est l'ensemble*

$I^{val}(a) = \{(val, e_1, \ldots, e_n) | e_1 \ldots e_n$ *un chemin de la racine à une feuille de l'arbre* $a\}$.

En particulier, $I^{val}(\Box) = \{val\}$. *L'interprétation d'un ensemble* G *de couples formés par une valeur et un arbre est par extension :*

$$I(G) = \bigcup_{(val,a)\in G} I^{val}(a).$$

Exemple 2 *En continuant l'exemple 1, l'interprétation de l'arbre a extrait de* G_t *:*



selon la valeur 0 est l'ensemble

$I^0(a) = \{(0,0,0,0), (0,0,0,1), (0,0,0,2), (0,0,1,0), (0,0,2,0), (0,1,1,1), (0,2,1,2), (0,2,2,1)\}$.

et

$I(G_t) = \{(0,0,0,0), (0,0,0,1), (0,0,0,2), (0,0,1,0), (0,0,2,0), (0,1,1,1), (0,2,1,2), (0,2,2,1), (1,1,0,0), (1,1,0,1), (1,1,0,2), (1,1,1,0), (1,1,2,0), (1,2,1,1), (2,2,0,0), (2,2,0,1), (2,2,0,2), (2,2,1,0), (2,2,2,0)\}$

Nous sommes alors en mesure de définir l'interprétation d'une base.

Définition 3 (interprétation d'une base) *La fonction* $(\cdot)^*$ *d'interprétation d'une base en un QCSP est définie par* $(Q = q_1 v_1 \ldots q_n v_n)$ *:*

$(top\_base)^* = \top$

$(bottom\_base)^* = \bot$

$(\langle Q \mid [G_{e_1}, \ldots, G_{e_m}]\rangle)^* =$
$Q \bigwedge_{e_k \in [e_1,\ldots,e_m]} ((v_{e_k}, v_1, \ldots, v_{e_k-1}) \in I(G_{e_k}))$

Exemple 3 *En continuant les exemples 1 et 2*

$(B)^* =$
$\exists x \exists y \forall z \exists t$
$((x \in I(G_x)) \wedge ((y,x) \in I(G_y)) \wedge ((t,x,y,z) \in I(G_t)))$

avec $x, y, z, t \in \{0,1,2\}$, $I(G_x) = \{0,1,2\}$ *et* $I(G_y) = \{0,1,2\}^2$.

3.3 Propriétés

À un QCSP donné, plusieurs bases peuvent correspondre dans le sens où l'interprétation de celles-ci a exactement les mêmes stratégies gagnantes que le QCSP, c'est ce que nous définissons comme la compatibilité.

Définition 4 (compatibilité d'une base) *Une base est compatible avec un QCSP si l'interprétation de celle-ci a exactement les mêmes stratégies gagnantes.*

Exemple 4 *En continuant les exemples 1, 2 et 3, la base* B *est compatible avec le QCSP* $\exists x \exists y \forall z \exists t (x = (y * z) + t)$ *avec* $x, y, z, t \in \{0,1,2\}$. *Si, par exemple, le couple* $(2, \Box)$ *de* G_x *est oté alors la base résultante n'est plus compatible car il lui manque deux des quatre stratégies gagnantes.*

Le théorème suivant est un théorème immédiat de complétude.

Théoreme 1 (complétude) *Pour tout QCSP il existe une base qui lui est compatible.*

Dans le cas d'un QCSP représentant un jeu à deux joueurs, une des questions les plus importantes, à chaque tour de jeu, pour le joueur existentiel est la suivante : « Que dois-je jouer pour être certain de gagner la partie ? ». Si une stratégie gagnante a été précalculée alors le joueur n'a qu'à suivre les prescriptions de celle-ci mais dans le cas où l'incertitude n'a pas été entièrement prise en compte et que la stratégie gagnante courante n'est plus d'actualité, le joueur doit recalculer entièrement une nouvelle stratégie gagnante et en payer le prix.

Définition 5 (prochain mouvement) *Instance :*
Un QCSP $q_1 v_1 \ldots q_n v_n \bigwedge_{c\in\mathcal{C}} c$ *avec* $v_1 \in D_1$, *...,* $v_n \in D_n$ *et une séquence de branchements* $v_1 = val_1$, *...,* $v_i = val_i$ *obtenue d'une solution au QCSP avec quant*$(v_1) = \exists$ *et* $val_1 \in D_1$, *...,* $val_i \in D_i$.
Question : Existe-t-il une stratégie gagnante pour un QCSP

$$q_{i+1} v_{i+1} \ldots q_n v_n \bigwedge_{c\in\mathcal{C}} c \wedge (v_1 = val_1) \wedge \ldots \wedge (v_{i-1} = val_{i-1}) \wedge (v_i = val'_i)$$

avec $v_{i+1} \in D_{i+1}$, *...,* $v_n \in D_n$, $val'_i \in D_i$, $val'_i \neq val_i$.

Clairement, dans le cas général, le problème du choix du prochain mouvement est encore un problème PSPACE-complet.

Nous introduisons la propriété d'optimalité pour une base qui va garantir que le problème du choix du prochain mouvement n'est plus un problème PSPACE-complet mais polynomial en temps par rapport à la taille de la base. Une base n'est dite *optimale* que si toute garde, qui est représentée sous forme d'arbre, associée à une valeur pour une variable, garantit que si la garde est passante pour les coups déjà joués (i.e. ils forment une branche de l'arbre ou encore, ce qui est équivalent, ils forment un n-uplet appartenant à l'interprétation de l'arbre selon la valeur) alors en jouant ce coup pour cette variable le joueur existentiel est sûr de continuer dans une stratégie gagnante.

Définition 6 (optimalité) *Soit une base* $B = \langle q_1 v_1 \dots q_n v_n \mid [G_{e_1}, \dots, G_{e_m}] \rangle$ *et* $(B)^* = q_1 v_1 \dots q_n v_n C_G$ *avec* $v_1 \in D_1$, …, $v_n \in D_n$. *La base est optimale si la propriété suivante est vérifiée. Pour tout* i, $i \in [1 \dots m]$, *soit* C_i *l'ensemble de contraintes* $\{(v_{e_k} = val_{e_k}) | 1 \leq k < i\}$ *telles que* $(val_{e_k}, val_{e_1}, \dots, val_{e_{k-1}}) \in I^{val_{e_k}}(a_{e_k})$, $(val_{e_k}, a_{e_k}) \in G_{e_k}$. *Alors pour tout couple* $(val, a) \in G_{e_i}$, $(val, val_{e_1}, \dots, val_{e_{i-1}}) \in I^{val}(a)$ *si et seulement si* $q_{e_i+1} v_{e_i+1} \dots q_n v_n (C_G \wedge (v^{e_i} = val) \wedge \bigwedge_{c \in C_i} c)$ *admet une stratégie gagnante.*

Exemple 5 *La figure 2 présente les ensembles de gardes* G_x^{opt}, G_y^{opt} *et* G_t^{opt} *de la base* $B^{opt} = \langle \exists x \exists y \forall z \exists t \mid [G_x^{opt}, G_y^{opt}, G_t^{opt}] \rangle$ *qui est optimale et compatible avec le QCSP :*

$$\exists x \exists y \forall z \exists t (x = (y * z) + t)$$

avec $x, y, z, t \in \{0, 1, 2\}$.

Nous explicitons ci-dessous l'optimalité de la base mais en faisant appel pour simplifier à la contrainte du QCSP qui lui est compatible.

- $i = 1$ *(i.e.* $v_{e_i} = x$*) alors* $C_i = \emptyset$ *et pour tout* $K \in \{0, 1, 2\}$, $K \in I^K(\square) = \{K\}$ *si et seulement si* $\exists y \forall z \exists t (x = (y * z) + t) \wedge (x = K)$, *avec* $y, z, t \in \{0, 1, 2\}$, *admet une stratégie gagnante.*
- $i = 2$ *(i.e.* $v_{e_i} = y$*) alors*
  - *pour tout* $K \in \{0, 1, 2\}$ $(K, \square) \in G_x^{opt}$, $I^0(T_0^y) = \{(0, 0), (0, 1), (0, 2)\}$ *et pour tout* $K \in \{0, 1, 2\}$, $(0, K) \in I^0(T_0^y)$ *si et seulement si* $\forall z \exists t (x = (y * z) + t) \wedge (y = 0) \wedge (x = K)$, *avec* $z, t \in \{0, 1, 2\}$, *admet une stratégie gagnante ;*
  - $(1, \square) \in G_x^{opt}$, $I^1(T_1^y) = \{(1, 1)\}$ *et* $(1, 1) \in I^1(T_1^y)$ *si et seulement si* $\forall z \exists t (x = (y * z) + t) \wedge (y = 1) \wedge (x = 1)$, *avec* $z, t \in \{0, 1, 2\}$, *admet une stratégie gagnante ;* $(1, 0) \notin I^1(T_1^y)$ *et* $\forall z \exists t (x = (y * z) + t) \wedge (y = 1) \wedge (x = 0)$, *avec* $z, t \in \{0, 1, 2\}$, *n'admet pas de stratégie gagnante ;* $(1, 2) \notin I^1(T_1^y)$ *et* $\forall z \exists t (x = (y * z) + t) \wedge (y = 1) \wedge (x = 2)$, *avec* $z, t \in \{0, 1, 2\}$, *n'admet pas de stratégie gagnante ;*
  - *pour* $y = 2$, *il n'y a pas de couple* $(2, T_2^y) \in G_y$ *et* $\exists x \forall z \exists t (x = (y * z) + t) \wedge (y = 2)$, *avec* $x, z, t \in \{0, 1, 2\}$, *n'admet pas de stratégie gagnante.*
- $i = 3$ *(i.e.* $v_{e_i} = t$*) alors (nous ne traitons que le cas* $t = 0$, *les autres cas sont similaires)*
  - $(2, \square) \in G_x^{opt}$, $(1, T_1^y) \in G_y^{opt}$ *avec* $(1, 2) \in I^1(T_1^y)$ *et* $(0, 2, 1, 2) \in I^0(T_0^t)$ *si et seulement si* $(x = (y * z) + t) \wedge (x = 2) \wedge (y = 1) \wedge (z = 2) \wedge (t = 0)$ *admet une stratégie gagnante ; il en est de même avec* $(0, 0, 0, 0)$, $(0, 0, 0, 1)$ *et* $(0, 0, 0, 2)$ *;*
  - *dans tous les autres cas* $(0, val_x, val_y, val_z) \notin I^0(T_0^t)$ *et* $(x = (y * z) + t) \wedge (x = val_x) \wedge (y = val_y) \wedge (z = val_z) \wedge (t = 0)$ *n'admet pas de stratégie gagnante.*

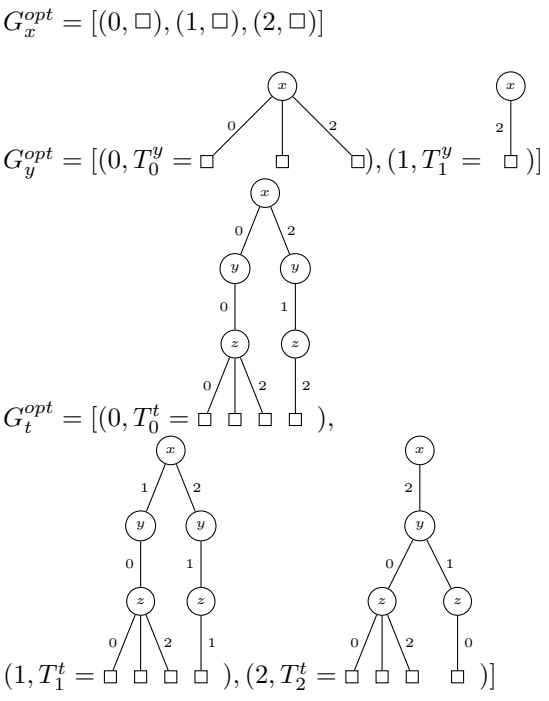


FIGURE 2 – Ensembles de gardes pour une base optimale.

Le théorème suivant montre tout l'intérêt de calculer une base optimale compatible avec un QCSP pour parcourir les stratégies gagnantes de ce dernier.

Théoreme 2 (prochain mouvement) *Le problème du choix du prochain mouvement pour l'interprétation d'une base optimale est polynomiale en temps par rapport à la taille de la base.*

Opérationnellement, à chaque tour de jeu, le joueur existentiel possédant une base optimale compatible avec la QCSP représentant le jeu, n'a qu'à vérifier que les coups déjà joués forment une branche de l'arbre de la garde associée au coup qu'il souhaite jouer.

4 Construction d'une base

Nous proposons deux algorithmes de construction d'une base dans le cadre d'un algorithme de recherche : le premier, récursif, est imbriqué dans ce dernier et reprend la structure inductive de la sémantique des QCSP, il est décrit dans la sous-section 4.1 ; le second, itératif, est un analyseur de trace d'exécution d'un solveur QCSP, il est décrit dans la sous-section 4.2.

4.1 Construction récursive d'une base

Algorithme 1 *rec_comp*

Entrée: Q : un lieur d'un QCSP
Entrée: C : un ensemble de contraintes d'un QCSP
Sortie: une base ou *top_base* ou *bottom_base*

```
si calcul_pointfixe_propagation(C) = failure alors
  retourner bottom_base
fin si
si vide(Q) alors retourner top_base fin si
qx_D ← tete(Q); listeValBase ← []; d ← D
tant que !vide(d) faire
  val ← tete(d); d ← queue(d)
  base ← rec_comp(queue(Q), C ∪ {x = val})
  si base = bottom_base&q = ∀ alors
    retourner bottom_base
  fin si
  listeValBase ← [(val, base)|listeValBase]
fin tant que
si vide(listeValBase) alors
  retourner bottom_base
fin si
si q = ∃ alors
  retourner ⊕∃(x_D, queue(Q), listeValBase)
sinon
  retourner ⊕∀(x_D, queue(Q), listeValBase)
fin si
```

La recherche d'une stratégie gagnante est définie de manière naturelle récursivement par la recherche simultanée sur les valeurs possibles de la variable ; l'algorithme 1 *rec_comp* de calcul d'une base compatible à partir d'un QCSP reproduit ce schéma. Il réalise en premier le calcul de point fixe de la propagation grâce à la fonction *calcul_pointfixe_propagation* sur l'ensemble des contraintes et retourne *bottom_base* si une contradiction est détectée ; si ce n'est pas le cas et que le lieur est vide alors *top_base* est renvoyé ; sinon, pour chaque valeur *val* du domaine de la variable x la plus externe du lieur, la contrainte $(x = val)$ est rajoutée aux contraintes et l'algorithme est appelé récursivement ; si la variable est associée à un quantificateur universel, il suffit qu'un des appels renvoient *bottom_base* pour que *bottom_base* soit renvoyé à son tour ; si la variable est associée à un quantificateur existentiel, il faut que tous les appels renvoient *bottom_base* pour que *bottom_base* soit renvoyé ; dans tous les autres cas, l'opérateur $\oplus_\exists$ ou $\oplus_\forall$ est invoqué pour combiner les bases entre-elles.

Algorithme 2 $\oplus_\exists$

Entrée: x_D : une variable et son domaine
Entrée: Q : un lieur
Entrée: l : une liste de paires (valeur,base)
Sortie: une base

```
si constantes(l) alors
  g ← cas_de_base(l)
  si vide(g) alors retourner bottom_base sinon
  retourner ⟨∃x_D Q | [g]⟩ fin si
sinon
  retourner ⟨∃x_D Q | [premiere(l)| ⊕ (x, l)]⟩
fin si
```

Algorithme 3 $\oplus_\forall$

Entrée: x_D : une variable et son domaine
Entrée: Q : un lieur
Entrée: l : une liste de paires (valeur,base)
Sortie: une base ou *top_base*

```
si constantes(l) alors
  retourner top_base
sinon
  retourner ⟨∀x_D Q | ⊕ (x, l)⟩
fin si
```

Les opérateurs $\oplus_\forall$ et $\oplus_\exists$ spécifiés respectivement par les algorithmes 2 et 3 opèrent ainsi : Pour le cas de base des deux algorithmes, la fonction *constantes* teste si la liste de couples passée en argument ne contient que des *top_base* ou des *bottom_base* pour second élément ; en cas de succès pour l'opérateur $\oplus_\forall$, il ne peut s'agir que de *top_base* dans le cas d'un bloc de quantificateurs universels le plus à l'intérieur et *top_base* est retourné ; en cas de succès pour l'opérateur $\oplus_\exists$, il ne peut s'agir que du quantificateur existentiel le plus interne et une base ne contenant que les valeurs associées aux *top_base* est construite grâce à la fonction *cas_de_base* définie par $cas\_de\_base(l) = \{(val, \square)|(val, top\_base) \in l\}$. En cas d'échec pour l'opérateur $\oplus_\exists$, la base renvoyée est construite en ajoutant au résultat de l'opérateur $\oplus$ la liste des valeurs associées aux bases grâce à la fonction *premiere* définie par $premiere(l) = \{(val, \square)|(val, a) \in l\}$. En cas d'échec pour l'opérateur $\oplus_\forall$, l'opérateur $\oplus$ spécifié par l'algorithme 4 est appliqué et le résultat est empaqueté dans une base qui est elle-même retournée puisque les

variables universelles quantifiées ne sont pas associées à des gardes.

Algorithme 4 $\oplus$

Entrée: x : une variable
Entrée: lvb : liste de paires (valeur,base)
Sortie: une liste de gardes

```
lg ← []
lvg ← extrait_gardes(lvb)
tant que !vide(lvg) faire
  dec_y ← decompose(lvg)
  lg ← [compose(x, 1°(dec_y))|lg]
  lvg ← 2°(dec_y)
fin tant que
retourner lg
```

Enfin l'opérateur $\oplus$ opère ainsi : la fonction *decompose* extrait de la liste lvg de paires (val, gardes), pour la variable y quantifiée existentiellement la plus externe du lieur, une paire constituée d'une liste de paires (val, liste des couples (valeur, arbre)) et d'une liste de paires (val, reste des gardes) ; la fonction *compose* construit pour y son ensemble de gardes en distribuant les arbres pour les différentes valeurs. Les fonctions 1° et 2° accèdent respectivement à la première et deuxième position d'un n-uplet ($n \geq 2$).

L'exemple suivant illustre le fonctionnement de l'algorithme *rec_comp*.

Exemple 6 *Nous calculons pour tout $val_x, val_y \in \{0,1,2\}$ la base $B_{val_x val_y}$ comme étant le résultat de l'appel :*

$$rec\_comp(\forall z \exists t, \{(x = (y * z) + t), (x = val_x), (y = val_y)\})$$

avec $z, t \in \{0,1,2\}$.

Nous obtenons les bases (selon $T^{val_x val_y}_{val_t}$) :

$$B_{00} = \langle \forall z \exists t | [[(0, T_0^{00} = \square_0\ \square\ \square_2)]] \rangle$$

$$B_{10} = \langle \forall z \exists t | [[(1, T_1^{10} = \square_0\ \square\ \square_2)]] \rangle$$

$$B_{20} = \langle \forall z \exists t | [[(2, T_2^{20} = \square_0\ \square\ \square_2)]] \rangle$$

$$B_{21} = \langle \forall z \exists t \mid [[(0, T_0^{21} = \square_2), (1, T_1^{21} = \square_1), (2, T_2^{21} = \square_0)]] \rangle$$

et pour toute autre combinaison, $B_{val_x val_y}$ = bottom_base.

L'exemple suivant illustre le partage des arbres qu'opère l'opérateur $\oplus$ montrant ainsi la distribution des arbres ; cet exemple illustre aussi qu'une base est de taille linéaire par rapport à un arbre contenant uniquement l'ensemble des scénarios appartenant à des stratégies gagnantes.

Exemple 7 *Le combinateur existentiel de bases est appliquée lors de l'appel*

$$rec\_comp(\exists y \forall z \exists t, \{(x = (y * z) + t), (x = 2)\})$$

avec $y, z, t \in \{0,1,2\}$, aux bases B_{20}, B_{21} et B_{22} qui représentent les bases pour les QCSP, respectivement, $\forall z \exists t((x = (y * z) + t) \wedge (x = 2) \wedge (y = 0))$, $\forall z \exists t((x = (y * z) + t) \wedge (x = 2) \wedge (y = 1))$, $\forall z \exists t((x = (y * z) + t) \wedge (x = 2) \wedge (y = 2))$.

$$\oplus_\exists(y, \forall z \exists t, [(0, B_{20}), (1, B_{21}), (2, B_{22})]) =$$
$$B_2 = \langle \exists y \forall z \exists t\ |[[(0, \square), (1, \square)],$$
$$[(0, T_0^{21}), (1, T_1^{21}), (2, T_2^{20}\ T_2^{21})]] \rangle$$

qui est une base compatible avec le QCSP

$$\exists y \forall z \exists t((x = (y * z) + t) \wedge (x = 2))$$

avec $y, z, t \in \{0,1,2\}$.

Le théorème suivant établit que non seulement l'algorithme *rec_comp* calcul à partir d'un QCSP une base compatible mais qu'en plus elle est optimale.

Théoreme 3 *Soit un QCSP QC. rec_comp(Q, C) retourne une base optimale et compatible avec le QCSP.*

4.2 Construction via un analyseur de trace

La section précédente a présenté une construction récursive de la base en considérant la définition inductive d'un QCSP et sa sémantique récursive mais en pratique il est souvent plus efficace de considérer, d'un côté, le solveur qui effectue la recherche et émet une trace d'exécution et, d'un autre côté, un analyseur de trace qui construit la base. Il y a deux raisons majeures à cette coopération entre deux processus via un flux :

- la gestion de la mémoire dans le solveur sous la forme d'une pile de retour-arrière qui doit, dans le cas d'un solveur incluant la construction de la base, stocker aussi l'état courant de celle-ci ;
- ternir compte des architectures modernes multi-cœurs.

L'analyseur de trace *it_comp* construit une base grâce à la trace d'exécution d'un solveur QCSP basé sur un algorithme de recherche. Cette trace est constituée d'une séquence de commandes de cinq types :

– `branche(x,val)` : la contrainte $(x = val)$ a été ajoutée à l'ensemble des contraintes, par branchement sur l'unique valeur possible du domaine lors de l'élimination du quantificateur pour les variables forcées par propagation ou par branchement lors d'un point de choix ;
– `echec` : le scénario partiel courant fait apparaître une contradiction dans l'ensemble des contraintes, celui-ci ne peut donc pas faire partie d'une stratégie gagnante ;
– `echec(`v_i`)` : le scénario courant $val_1, \ldots, val_{i-1}$ pour les variables $v_1, \ldots v_{i-1}$ est tel que le QCSP $\forall v_i q_{i+1} v_{i+1} \ldots q_n v_n (C \wedge \bigwedge_{1 \leq k < i} (v_k = val_k))$ est détecté comme sans stratégie gagnante ;
– `succes` : le scénario courant vérifie toutes les contraintes ;
– `succes_global` : une stratégie gagnante est calculée.

La trace t est un flux dans lequel le solveur enfile[5] les commandes et que l'analyseur défile grâce à la fonction *defile* ou consulte grâce à la fonction *consulte*. La variable s (resp. p) représente les quantificateurs sur lesquels le solveur n'a pas encore (resp. a déjà) branché ; la variable sc représente le scénario courant ; la variable st représente le stock de scénarios gagnants ; la variable lg représente les gardes déjà calculées ; la variable c représente une commande ; la variable *etat* représente l'état de l'algorithme. L'état *descente* représente la descente dans l'arbre de recherche (i.e. la construction du scénario) et n'est modifié en état *backtrack* que lorsqu'un succès est rencontré (et le scénario est alors stocké dans la liste des scénarios gagnants) ou lorsqu'un échec est rencontré. L'état *backtrack* représente la remontée dans l'arbre de recherche à la recherche du point de choix précédent et n'est modifié que lorsque celui-ci est trouvé. Dans l'état de *descente* : la commande $branche(x, val)$ correspond à l'ajout de la contrainte $(x = val)$ dans l'ensemble des contraintes et entraîne l'ajout de val dans le scénario et du glissement du quantificateur de s à p. Dans l'état de *backtrack* : la commande $branche(x, val)$ permet de retrouver le point de choix en faisant glisser les quantificateurs de p à s jusqu'à ce que la variable de branchement corresponde à la variable associée au quantificateur ; la commande *succes_global* correspond à l'insertion d'une stratégie gagnante dans la liste des gardes partielles via la fonction *insere* ; la commande $echec(u)$ élimine des scénarios gagnants ceux qui associent, pour le même scénario partiel que le courant, d'autres valeurs à la variable universelle u. Les fonctions 3° et 4° accèdent respectivement à la troisième et quatrième position d'un 4-uplet.

5. les fonctions $defile$, $enfile$ et *consulte* sont les opérations classiques sur une file.

Algorithme 5 *it_comp*

Entrée: Q : un lieur d'un QCSP
Entrée: t : une trace d'exécution d'un solveur QCSP
Sortie: une base

```
etat = descente; s = Q; p = ε; sc = []; st = []
lg = gardes_vides(Q)
tant que !vide(t) faire
 c ← consulte(t)
 si etat = descente alors
  defile(t)
  selon c faire
   cas branche(x, val) :
   s ← queue(s); p ← [tete(s)|p]; sc ← [val|sc]
   cas succes :
   etat = backtrack; st ← [sc|st]
   cas echec :
   etat = backtrack
  fin selon
 sinon
  selon c faire
   cas branche(x, val) :
   p ← queue(p); s ← [tete(p)|s]
   si x = 1°(tete(sc)) alors
    etat = descente
   fin si
   sc ← queue(sc)
   cas succes_global :
   defile(t)
   lg ← insere(st, lg)
   cas echec(u) :
   defile(t)
   q ← destocke(u, s, p, sc, st)
   s ← 1°(q); p ← 2°(q); sc ← 3°(q); st ← 4°(q)
  fin selon
 fin si
fin tant que
retourner ⟨Q | lg⟩
```

Exemple 8 *La figure 3 présente les ensembles de gardes G'_x, G'_y et G'_t de la base $B' = \langle \exists x \exists y \forall z \exists t \mid [G'_x, G'_y, G'_t] \rangle$, qui est calculée par l'analyseur de trace it_comp après avoir intégré les trois premières stratégies gagnantes pour le QCSP : $\exists x \exists y \forall z \exists t (x = (y * z) + t)$ avec $x, y, z, t \in \{0, 1, 2\}$.*

L'analyseur de trace it_comp est alors dans l'état backtrack avec les quantificateurs sur lesquels le solveur n'a pas encore branché $s = []$ (respectivement, a déjà branché $p = [\exists t, \forall z, \exists y, \exists x]$), le scénario courant

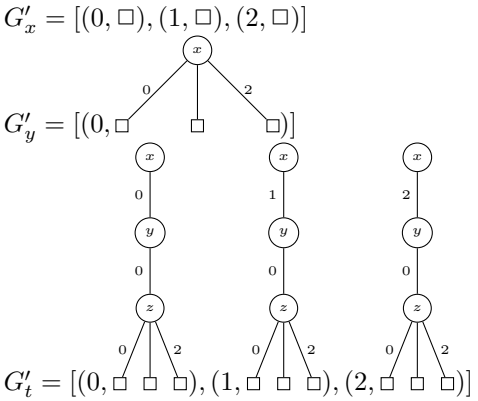
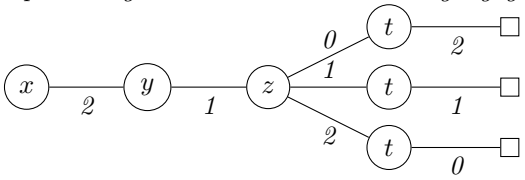


FIGURE 3 – Ensembles de gardes pour une base intermédaire.

inversé $[2,2,0,2]$ [6] *et la trace :*

$[branche(y,1), branch(z,0), branche(t,0), echec,$
$branche(t,1), echec, branche(t,2), succes,$
$branche(z,1), branche(t,0), echec, branche(t,1),$
$succes, branche(t,2), echec, branche(z,2),$
$branche(t,0), succes, branche(t,1), echec,$
$branche(t,2), echec, succes\_global, branche(y,2),$
$branche(z,0), branche(t,0), echec, branche(t,1),$
$echec, branche(t,2), succes, branche(z,1),$
$branche(t,0), succes, branche(t,1), echec,$
$branche(t,2), echec, branche(z,2), branche(t,0),$
$echec, branche(t,1), echec, branche(t,2),$
$echec, echec(z)]$

Après intégration de la dernière stratégie gagnante :



nous obtenons la base de la figure 2 de l'exemple 5.

Le théorème suivant établit que non seulement l'algorithme *it_comp* calcule à partir d'un QCSP une base compatible mais qu'en plus elle est optimale.

Théoreme 4 *Soit un QCSP QC et T la trace obtenue pour un solveur QCSP basé sur un algorithme de recherche. it_comp(Q,T) retourne une base optimale et compatible avec le QCSP.*

4.3 Complémentarité des deux approches

Les deux approches décrites dans les deux sous-sections précédentes sont complémentaires dans une architecture multicœurs/multiprocessus : à des couples

6. i.e. il faut lire $(t,2),(z,2),(y,0),(x,2)$.

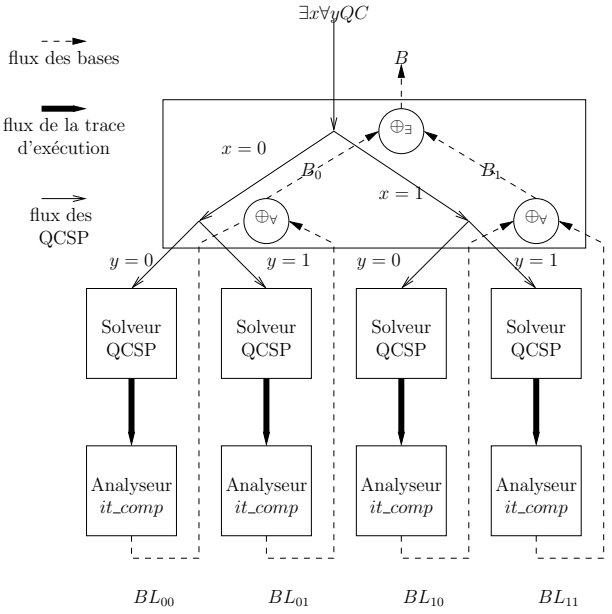


FIGURE 4 – Exemple d'architecture à 9 processus combinant les opérateurs $\oplus_\exists$ et $\oplus_\forall$ et l'analyseur de trace *it_comp*.

(algorithme de recherche pour QCSP, analyseur de trace *it_comp*) travaillant en parallèle, la tâche est confiée de calculer des bases pour des sous-problèmes tandis qu'un serveur combine ces bases grâce aux opérateurs $\oplus_\exists$ et $\oplus_\forall$.

Exemple 9 *La figure 4 présente pour un QCSP $\exists x \forall y QC$ avec $x,y \in \{0,1\}$, un exemple d'architecture avec un serveur réalisant la recherche (ici les branchements sur les valeurs des domaines des variables x et y) et appliquant aux bases B_{00}, B_{01} B_{10} et B_{11} l'opérateur $\oplus_\forall$ puis aux bases résultats B_0 et B_1 l'opérateur $\oplus_\exists$ pour obtenir une base finale B compatible avec le QCSP et optimale ; les bases B_{00}, B_{01} B_{10} et B_{11} ayant été obtenues dans quatre processus différents par une collaboration entre un algorithme de recherche et l'analyseur de trace it_comp pour les QCSP, respectivement, $Q(C \wedge (x=0) \wedge (y=0))$, $Q(C \wedge (x=0) \wedge (y=1))$, $Q(C \wedge (x=1) \wedge (y=0))$ et $Q(C \wedge (x=1) \wedge (y=1))$.*

5 Conclusion

Nous avons proposé dans cet article un cadre formel pour la compilation des QCSP : les bases. Grâce à nos deux algorithmes *rec_comp* et *it_comp*, nous sommes à même de nous adapter à quasiment toute architecture basée sur un algorithme de recherche. Ces algorithmes permettent pour des QCSP de générer des bases optimales qui leur soient compa-

tibles. Nous disposons d'implantations en Prolog pour les algorithmes décrits dans cet article disponibles à l'adresse `http://www.info.univ-angers.fr/pub/stephan/Research/Download.html` et projetons de l'insérer dans sa version itérative *it_comp* à notre solveur QCSP développé dans l'environnement générique pour le développement de systèmes de contraintes Gecode [14].

Dans le cadre d'un algorithme de décision pour les QCSP, lorsque le solveur répond qu'il y a ou n'y a pas de stratégie gagnante, rien ne permet de le vérifier. Un certificat est une information, qui peut être une stratégie gagnante mais pas nécessairement, qui permet de vérifier que l'algorithme à bien décider du problème. Nous n'avons pas pu, faute de place, traiter les certificats pour les QCSP : notre formalisme les inclut comme un cas particulier ; l'interprétation des arbres sous la forme de contraintes tables permet de vérifier un tel certificat vis-à-vis d'un QCSP par la résolution d'un problème co-NP-complet (rejoignant ici la complexité de la vérification d'une politique pour une QBF [3]).

Références

[1] F. Bacchus and K. Stergiou. Solution directed backjumping for qcsp. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, pages 148–163, 2007.

[2] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E.Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of the international conference on Computer-aided design (ICCAD'93)*, 1993.

[3] M. Benedetti. Extracting Certificates from Quantified Boolean Formulas. In *Proceedings of 9th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 47–53, 2005.

[4] M. Benedetti, A. Lallouet, and J. Vautard. Reusing csp propagators for qcsps. In *Recent Advances in Constraints, 11th Annual ERCIM International, Workshop on Constraint Solving and Contraint Logic Programming (CSCLP'06)*, pages 63–77, 2006.

[5] L. Bordeaux and E. Monfroy. Beyond NP : Arc-Consistency for Quantified Constraints. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*, pages 371–386, 2002.

[6] M. Cadoli and F.M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3-4) :137–150, 1997.

[7] H. Chen. The Computational Complexity of Quantified Constraint Satisfaction, PhD thesis, cornell university, 2004.

[8] S. Coste-Marquis, D. Le Berre, F. Letombe, and P. Marquis. Complexity Results for Quantified Boolean Formulae Based on Complete Propositional Languages. *Journal on Satisfiability, Boolean Modeling and Computation*, 1 :61–88, 2006.

[9] A. Darwiche and P. Marquis. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research*, 17 :229–264, 2002.

[10] H. Fargier and P. Marquis. On the Use of Partially Ordered Decision Graphs in Knowledge Compilation and Quantified Boolean Formulae. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI'06)*, 2006.

[11] I. Gent, P. Nightingale, and K. Stergiou. Qcsp-solve : A solver for quantified constraint satisfaction problems. In *Proceedings of 9th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 2005.

[12] I.P. Gent, P. Nightingale, and A. Rowley. Encoding Quantified CSPs as Quantified Boolean Formulae. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, pages 176–180, 2004.

[13] A.K. Mackworth. Consistency in networks of relations. In *Artificial Intelligence*, volume 8, pages 99–118, 1977.

[14] C. Schulte and G. Tack. Views and Iterators for Generic Constraint Implementations. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, pages 817–821, 2005.

[15] I. Stéphan and B. Da Mota. Base littérale et certificat pour les formules booléennes quantifiées. In *Actes des Troisièmes Journées Francophones de Programmation par Contraintes (JFPC'08)*, pages 307–316, 2008.

[16] I. Stéphan and B. Da Mota. A unified framework for Certificate and Compilation for QBF. In *Proceedings of the 3rd Indian Conference on Logic and its Applications*, pages 210–223, 2009.

[17] G. Verger and C. Bessière. BlockSolve : une approche bottom-up des QCSP. In *Actes des Deuxièmes Journées Francophones de Programmation par Contraintes (JFPC'06)*, pages 337–345, 2006.

The Complexity of Valued Constraint Satisfaction Problems in a Nutshell

Johan Thapper[1] *

[1] École Polytechnique, Laboratoire d'Informatique (LIX), 91128 Palaiseau, France
thapper@lix.polytechnique.fr

Abstract

The valued constraint satisfaction problem was introduced by Schiex et al. [23] as a unifying framework for studying constraint programming with soft constraints. A systematic worst-case complexity theoretical investigation of this problem was initiated by Cohen et al. [4], building on ideas from the successful classification programme for the ordinary constraint satisfaction problem. In addition to the decision problem for constraint satisfaction, this framework also captures problems as varied as Max CSP and integer programming with bounded domains.

This paper is intended to give a quick introduction to the questions, the main results, and the current state of the complexity classification of valued constraint satisfaction problems. Two special cases are looked at in some detail : the classification for the Boolean domain and the less well-understood case of Max CSP. Some recent results for general constraint languages are also reviewed, as well as the connection to the very active study of approximation algorithms for Max CSP.

1 Introduction

The valued constraint satisfaction problem was introduced by Schiex et al. [23] as a unifying framework for studying constraint programming with soft constraints. It has also proved to be a convenient framework for studying the complexity of various optimisation variations of constraint satisfaction problems. This paper gives a quick introduction to the type of questions studied in this area by reviewing the main results for the Boolean domain and for Max CSP. The focus is on the *constraint language* parameterisation, but other complexity questions have also been considered, see for example [6].

*The author has received funding from the ERC FP7/2007-2013 Grant Agreement no. 257039.

In the original definition by Schiex et al. [23], a VCSP is defined over a *valuation structure* ; a totally ordered set together with an *aggregation operator* satisfying certain basic properties. Here we will exclusively consider the valuation structure $\overline{\mathbb{Q}}_{\geq 0}$; the extended non-negative rational numbers with the natural aggregation operator $+$, where $x + \infty = \infty$ for all x.

Definition 1.1 *A VCSP-instance consists of a triple (V, D, C), where*
- *V is a finite set of* variables *;*
- *D is a finite set of* domain elements *; and*
- *C is a finite set of constraints $c = \langle \bar{x}, f \rangle$, where $\bar{x}$,* the scope*, is a tuple of variables from V and f is a function from $D^{|\bar{x}|}$ to $\overline{\mathbb{Q}}_{\geq 0}$.*

A constraint with a function f is called *soft* if f does not take any infinite value. It is called *crisp* if f takes values in $\{0, \infty\}$. An *assignment* to a VCSP-instance $\mathcal{I}$ is a function $\sigma : V \to D$. The *measure*, $m_{\mathcal{I}}(\sigma)$, of σ is the total aggregated cost of the constraints of $\mathcal{I}$,

$$m_{\mathcal{I}}(\sigma) := \sum_{\langle \bar{x}, f \rangle \in C} f(\sigma(\bar{x})),$$

where σ is applied component-wise to $\bar{x}$. The measure is sometimes also denoted by $Cost_{\mathcal{I}}(\sigma)$. The objective is to *minimise* the measure over all assignments.

A *(valued) constraint language* $\mathcal{F}$ is a set of functions $f : D^k \to \overline{\mathbb{Q}}_{\geq 0}$. The problem VCSP($\mathcal{F}$) is the set of VCSP-instances in which the cost functions come from $\mathcal{F}$. If there exists an algorithm that solves VCSP($\mathcal{F}$) in polynomial time, then $\mathcal{F}$ is said to be *tractable*. If there is a polynomial-time reduction to VCSP($\mathcal{F}$) from some NP-hard problem, $\mathcal{F}$ is said to be NP-hard.

Example 1 (CSP) *A standard constraint satisfaction problem is given by a set of relations applied to tuples of variables : $R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m)$. This can be*

expressed as a VCSP using the following translation : For each k-ary R_i, define the function $f_{R_i} : D^k \to \overline{\mathbb{Q}}_{\geq 0}$ by $f_{R_i}(\bar{t}) = 0$ if $\bar{t} \in R_i$ and $f_{R_i}(\bar{t}) = \infty$ otherwise. The VCSP instance is then given by the constraints $\langle \bar{x}_i, f_{R_i} \rangle$ for $i \leq m$. An assignment $\sigma : V \to D$ satisfies the original instance iff $\sum_i f_{R_i}(\sigma(\bar{x}_i)) = 0$.

Example 2 (Max CSP) *One of the most well-studied optimisation variations of CSP is the Max CSP problem : Rather than asking whether an instance is satisfiable, one wants to maximise the number of satisfied constraints. Although this is a maximisation problem, it can for some purposes be modelled as a VCSP*[1]*. In this case the translation takes a relation R to the function f_R such that $f_R(\bar{t}) = 0$ if $\bar{t} \in R$ and $f_R(\bar{t}) = 1$ otherwise. Hence, the degree to which non-satisfying assignments are penalised depends on the number of unsatisfied constraints.*

Example 3 (Min Ones) *Given a CSP-instance over the Boolean domain, the problem Min Ones is to either decide that the instance is unsatisfiable, or to return a satisfying assignment with as few variables as possible set to the value 1 (true). The weighted version of this problem can be expressed as a VCSP by taking the functions used in Example 1 for expressing the crisp constraints together with the unary function $f(t) = t$ for $t = 0, 1$.*

The last example can be seen as the result of adding a linear objective function to a CSP. A generalisation of this idea to larger domains leads to the definition of *the maximum solution problem* [12, 14, 15]. The generalised problem can also be expressed as a VCSP. It models a variety of problems including integer programming over bounded domains.

2 The Boolean Case

A relation is 0-*valid* (resp. *1-valid*) if it contains the all-0 (resp. all-1) tuple. A Boolean relation is 2-*monotone* if it can be written on the form $R(x_1, \ldots, x_p, y_1, \ldots, y_q) \equiv (x_1 \wedge \cdots \wedge x_p) \vee (\neg y_1 \wedge \cdots \wedge \neg y_q)$. A set of Boolean relations Γ is said to be 0-valid (1-valid, 2-monotone) if all its relations are 0-valid (1-valid, 2-monotone).

The following is good example of the type of result one encounters in this area :

Theorem 2.1 (Creignou [7]) *Let Γ be a set of Boolean relations. Then Max CSP(Γ) can be solved in polynomial time if Γ is* 0-valid, 1-valid, *or if Γ is* 2-monotone. *Otherwise the problem is* NP*-hard.*

1. The naturally corresponding minimisation problem is equivalent with respect to exact optimisation, but *not* with respect to approximation.

That is, for some type of cost functions (finite-valued, $\{0, 1\}$-valued, etc.,) one looks for a classification of the complexity of VCSP into a small number of complexity classes. The complexity classes vary depending on the tools available. For this reason, some classifications are made up to approximation preserving reductions, while others settle for distinguishing between tractability and NP-hardness.

Khanna et al. [16] gives classifications for the Boolean domain cases of Max CSP, Min CSP, Max Ones, and Min Ones, and their weighted counterparts (see also [8]). Their main tool is called *strict implementations*, a type of gadget that preserves certain approximation properties.

Cohen et al. [4] initiated the systematic study of the complexity of VCSPs. They introduced several concepts inspired by the study of ordinary CSPs. In place of strict implementations they define *expressibility* modelled after *pp-definability*. This provides more flexibility but the reductions involved are now polynomial-time many-one reductions and do not allow the distinction between various approximation classes.

They also introduce *multimorphisms*; a notion roughly corresponding to *polymorphisms* of relational structures used in the study of ordinary CSPs.

Let $f : D^m \to \overline{\mathbb{Q}}_{\geq 0}$ be a cost function. Let $\sqcap, \sqcup : D^2 \to D$ be two binary operations on D. Then $\langle \sqcap, \sqcup \rangle$ is called *a (binary) multimorphism of f* if, for any two m-tuples, a and b,

$$f(a \sqcap b) + f(a \sqcup b) \leq f(a) + f(b),$$

where $\sqcap$ and $\sqcup$ are applied component-wise. Multimorphisms of higher arity are defined analogously. If $\langle \sqcap, \sqcup \rangle$ is a multimorphism of every cost function in a valued constraint language $\mathcal{F}$, then $\langle \sqcap, \sqcup \rangle$ is said to be *a multimorphism of $\mathcal{F}$*.

Many known classes of tractable problems and polynomial-time algorithms can be directly linked to the fact that a valued constraint language has some particular multimorphism.

Example 4 *Crisp so-called max-closed languages are known to be solvable by arc-consistency from the study of standard CSPs. These are generalised by valued constraint languages with the multimorphism $\langle \max, \max \rangle$. All such languages are tractable.*

Example 5 *The condition for tractability in Theorem 2.1 can be reformulated in terms of multimorphisms : R is* 0*-valid (*1*-valid) iff $f_R(0, \ldots, 0) = 0$ ($f_R(1, \ldots, 1) = 0$) iff f_R has the unary multimorphism $\langle 0 \rangle$ ($\langle 1 \rangle$), where* 0 *and* 1 *denotes the constant unary functions. Furthermore, a relation R is* 2*-monotone iff $1 - f_R$ has the multimorphism $\langle \min, \max \rangle$.*

Armed with expressibility and multimorphisms, Cohen et al. classify *all* tractable valued constraint languages on the Boolean domain into 8 cases based on the multimorphisms they possess. These are given by :

- $\langle 0\rangle$, $\langle 1\rangle$;
- $\langle \min, \max\rangle$, $\langle \min, \min\rangle$, $\langle \max, \max\rangle$; and
- $\langle \mathrm{mj}, \mathrm{mj}, \mathrm{mj}\rangle$, $\langle \mathrm{mn}, \mathrm{mn}, \mathrm{mn}\rangle$, $\langle \mathrm{mj}, \mathrm{mj}, \mathrm{mn}\rangle$,

where mj (mn) denotes the *majority* (*minority*) operation. All other valued constraint languages on the Boolean domain are NP-hard.

Note that this classification contains Schaefer's classification of Boolean CSPs [22], the Boolean Max CSP, Min CSP, Max Ones, and Min Ones problems considered by Creignou and Khanna et al., as well as problems with mixed cost functions. On the other hand, due to the nature of the reductions involved, this classification does not reproduce the various approximation classes distinguished in [16].

3 Max CSP

The two first conditions in Theorem 2.1 seem trivial : If Γ is 0-valid, then *any* instance of Max CSP(Γ) can be mapped to an equivalent instance of a language on a domain with a single element. Informally Γ is a called *core* if it cannot be reduced to a language on a smaller domain in this simple fashion.

Theorem 2.1 gives a classification for Max CSP on a Boolean domain. The three-element domain case was classified in [11]. The case when Γ contains all unary relations was classified in [9]. The proofs of these results use cleverly conducted computer-aided searches in conjunction with the strict implementations to obtain dichotomies between tractable and APX-hard constraint languages.

Let $\mathcal{F}$ be a set of $\{0,1\}$-valued functions. All of the mentioned results for Max CSP (including Creignou's original result for the Boolean domain) can be stated for VCSP on the following form :

Assuming that $\mathcal{F}$ is a core, $\mathcal{F}$ is tractable iff it has the multimorphism $\langle \min, \max\rangle$ for some order on the domain of $\mathcal{F}$. Otherwise $\mathcal{F}$ is NP*-hard.*

Given a total order on the domain, a cost function f with a multimorphism $\langle \min, \max\rangle$ is called *submodular (on the fixed order)*, where min and max are taken with respect to the order. Submodular functions appear as an important class of tractable valued constraint languages for minimisation.

More generally, one can define submodularity over an arbitrary *lattice* on the domain. The meet and join of the lattice then become the two operations of a multimorphism. The realisation that this defines important tractable subclasses of Max CSP was made in [3], where it was also conjectured that, the only source of tractability for a core $\mathcal{F}$ is submodularity on a lattice. This conjecture was disproved in [13] where a classification for the four-element domain case showed that there are tractable $\{0,1\}$-valued constraint languages that are not submodular with respect to any lattice.

Tractability for more general soft constraint languages based on generalisations of submodularity are considered in [2, 24].

4 Recent Developments

While multimorphisms (and various other concepts mimicking the universal-algebraic study of CSPs) have been around for a while, a completely satisfactory theory has been lacking. This is remedied by the introduction of *weighted polymorphisms* in [5], which are shown to completely determine the complexity of a general valued constraint language. The paper also defines a *Galois connection* between valued constraint languages and sets of weighted polymorphisms.

Another recent result is the classification of valued constraint languages containing all (soft) unary functions [18]. Interestingly, this result does *not* use the advanced algebraic machinery of weighted polymorphisms, but is instead based on a graph-representation of partial multimorphisms.

Parallel to the development of algebraic tools for dealing with VCSP, a different community has made immense progress on the approximability of Max CSP and other CSP-related optimisation problems. Goemans and Williamson [10] introduced rounding of semidefinite programming (SDP) relaxations as a basis for approximation algorithms. Their approximation algorithm for Max cut (Max CSP($\{\neq\}$) on a Boolean domain) achieves a constant approximation ratio of 0.87856, beating the trivial ratio 0.5 (achievable by taking a random assignment) that was up until then the best known. The Goemans and Williamson approximation ratio for Max cut has been matched by an upper bound, i.e., a hardness-result showing that, under the *unique games conjecture (UGC)* [17], the ratio obtained by the algorithm is the best possible.

Building on these ideas, Raghavendra [20] eventually managed to develop SDP-based algorithms for *every* Max CSP-problem, with constant approximation ratios that are *optimal*, provided that the UGC holds. For a tractable constraint language, Raghavendra's algorithm should achieve a ratio of 1 and thereby provide the positive part of a complete classification for Max CSP. There are however various technical complications involved. For one, the SDP relaxations can only be solved optimally up to an (arbitrarily small) constant. Furthermore, it is not clear how

to determine the ratio achieved by an algorithm given a fixed constraint languages. Raghavendra and Steurer [21] show how one may in principle approximate (with any desired precision) the ratio for an arbitrarily given constraint language, but their algorithm is doubly exponential in the domain size.

Raghavendra's result on SDP relaxations, recent work on LP-relaxations [19,24], and on *robust approximation* [1,19] seem to be bringing closer together the interests of the two communities working on, respectively, the approximation of CSPs and the classification projects of CSPs and VCSPs.

Références

[1] Libor Barto and Marcin Kozik. Robust Satisfiability of Constraint Satisfaction Problems. In *Proceedings of STOC-2012*, 2012.

[2] David Cohen, Martin Cooper, and Peter Jeavons. Generalising submodularity and Horn clauses : Tractable optimization problems defined by tournament pair multimorphisms. *Theor. Comput. Sci.*, 401(1-3) :36–51, 2008.

[3] David Cohen, Martin Cooper, Peter Jeavons, and Andrei Krokhin. Supermodular functions and the complexity of MAX CSP. *Discrete Appl. Math.*, 149(1-3) :53–72, 2005.

[4] David Cohen, Martin Cooper, Peter Jeavons, and Andrei Krokhin. The complexity of soft constraint satisfaction. *Artificial Intelligence*, 170(11) :983–1016, 2006.

[5] David Cohen, Páidí Creed, Peter Jeavons, and Stanislav Živný. An algebraic theory of complexity for valued constraints : Establishing a galois connection. In *Proceedings of MFCS-2011*, pages 231–242, 2011.

[6] Martin C. Cooper and Stanislav Živný Tractable triangles. In *Proceedings of CP-2011*, pages 195–209, 2011.

[7] Nadia Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *J. Comput. Syst. Sci.*, 51(3) :511–522, 1995.

[8] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of boolean constraint satisfaction problems*. SIAM, Philadelphia, PA, USA, 2001.

[9] Vladimir Deineko, Peter Jonsson, Mikael Klasson, and Andrei Krokhin. The approximability of MAX CSP with fixed-value constraints. *J. ACM*, 55(4) :1–37, 2008.

[10] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42 :1115–1145, 1995.

[11] Peter Jonsson, Mikael Klasson, and Andrei Krokhin. The approximability of three-valued MAX CSP. *SIAM J. Comput.*, 35(6) :1329–1349, 2006.

[12] Peter Jonsson, Fredrik Kuivinen, and Gustav Nordh. MAX ONES generalized to larger domains. *SIAM J. Comput.*, 38(1) :329–365, 2008.

[13] Peter Jonsson, Fredrik Kuivinen, and Johan Thapper. Min CSP on four elements : Moving beyond submodularity. In *Proceedings of CP-2011*, pages 438–453, 2011.

[14] Peter Jonsson and Gustav Nordh. Introduction to the maximum solution problem. In Nadia Creignou, Phokion G. Kolaitis, and Heribert Vollmer, editors, *Complexity of Constraints – An Overview of Current Research*, volume 5250 of *LNCS*, pages 255–282. Springer, 2008.

[15] Peter Jonsson and Johan Thapper. Approximability of the maximum solution problem for certain families of algebras. In *Proceedings of CSR-2009*, pages 215–226, 2009.

[16] Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6) :1863–1920, 2000.

[17] Subhash Khot. On the unique games conjecture (invited survey). *Annual IEEE Conference on Computational Complexity*, pages 99–121, 2010.

[18] Vladimir Kolmogorov and Stanislav Živný. The complexity of conservative valued CSPs. In *Proceedings of SODA-2012*, pages 750–759, 2012.

[19] Gábor Kun, Ryan O'Donnell, Suguru Tamaki, Yuichi Yoshida, and Yuan Zhou. Linear programming, width-1 CSPs, and robust satisfaction. In *Proceedings of ITCS-2012*, pages 484–495, 2012.

[20] Prasad Raghavendra. Optimal algorithms and inapproximability results for every CSP ? In *Proceedings of STOC-2008*, pages 245–254, 2008.

[21] Prasad Raghavendra and David Steurer. How to round any CSP. In *Proceedings of FOCS-2009*, pages 586–594, 2009.

[22] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of STOC-1978*, pages 216–226, 1978.

[23] Thomas Schiex, Helene Fargier, and Gerard Verfaillie. Valued constraint satisfaction problems : Easy and hard problems. In *Proceedings of IJCAI-1995*, pages 631–637, 1995.

[24] Johan Thapper and Stanislav Živný. The power of linear programming for valued CSPs. arXiv :1204.1079 [cs.CC], 2012.

Extraction de motifs sous contraintes souples de seuil

Willy Ugarte Patrice Boizumault Samir Loudni Bruno Crémilleux
GREYC (CNRS UMR 6072) – Université de Caen Basse-Normandie
Campus II, Côte de Nacre, 14000 Caen - France
prenom.nom@unicaen.fr

Résumé

Lors de l'extraction de motifs sous contraintes, les contraintes sur des mesures (fréquence, taille des motifs) sont parmi les plus utilisées. Dans la pratique, fixer une valeur de seuil pour une mesure est un problème difficile et le choix d'un seuil relève souvent de l'arbitraire. De plus, la rigidité du cadre actuel fait que des motifs pouvant s'avérer en réalité intéressants ne sont pas extraits car ratant de peu le seuil. Dans cet article, nous montrons comment les contraintes souples de seuil peuvent être mises en œuvre dans un extracteur de motifs (basé-CSP). Nous montrons la pertinence et la faisabilité de notre approche au travers d'une application en chémoinformatique portant sur la découverte de fragments moléculaires toxicophores.

Abstract

Constraint-based pattern discovery is at the core of numerous data mining tasks. Patterns are extracted with respect to a given set of constraints (frequency, closedness, size, etc). In practice, many constraints require threshold values whose choice is often arbitrary. This difficulty is even harder when several thresholds are required and have to be combined. Moreover, patterns barely missing a threshold will not be extracted even if they may be relevant. In this paper, by using CP we propose a method to integrate soft threshold constraints into the pattern discovery process. We show the relevance and the efficiency of our approach through a case study in chemoinformatics for discovering toxicophores.

1 Introduction

L'extraction de connaissances dans les bases de données, aussi appelée "fouille de données", a pour but la découverte automatique d'informations nouvelles et interprétables en connaissances utiles. Une étape centrale de ce processus est l'extraction de motifs sous contraintes, les contraintes permettant de cibler la recherche des informations à extraire suivant les centres d'intérêt de l'utilisateur. Cette forme d'extraction s'est essentiellement développée dans le cas des motifs locaux, ces derniers capturant les phénomènes valides sur une portion de la base de données [8, 12]. Les contraintes sur des mesures telles que la fréquence (nombre d'occurrences d'un motif dans la base de données) ou encore la taille (nombre d'attributs composant un motif) sont parmi les contraintes les plus utilisées.

Dans la pratique, fixer une valeur de seuil pour une mesure est un problème difficile et le choix d'un seuil relève souvent de l'arbitraire. De plus, la rigidité du cadre actuel fait que des motifs pouvant s'avérer en réalité intéressants (i.e.les motifs dont la mesure est proche du seuil mais ne le satisfait pas) ne sont pas examinés. Ce problème est encore plus ardu lorsque plusieurs seuils doivent être fixés simultanément dans une même requête. Une première tentative pour résoudre ce problème, dans le cas des extracteurs de motifs locaux, a été proposée dans [4, 5].

Récemment, plusieurs travaux ont montré l'apport de la Programmation Par Contraintes (PPC) pour l'extraction de motifs locaux [13, 17] ou n-aires [10, 11]. Le point commun de l'ensemble de ces travaux est de modéliser les problèmes d'extraction de motifs, qu'ils soient locaux ou n-aires, sous forme de Problèmes de Satisfaction de Contraintes (CSP). La résolution de ce CSP produit l'ensemble complet des motifs satisfaisant toutes les contraintes.

Dans cet article, nous montrons comment les contraintes souples de seuil peuvent être mises en œuvre dans un extracteur de motifs (basé-CSP) en utilisant les travaux existants en PPC sur la relaxa-

tion de contraintes et les préférences entre solutions. Puis la pertinence et la faisabilité de notre approche sont illustrées au travers d'une application en chémoinformatique portant sur la découverte de fragments moléculaires toxicophores.

Pour cela, à chaque contrainte souple de seuil, est associée une sémantique de violation permettant de mesurer l'écart au seuil. Puis, nous montrons que toute requête incluant des contraintes souples de seuil peut être transformée en une requête équivalente, uniquement constituée de contraintes dures, pouvant être résolues par un solveur de CSP. Nous nous intéressons alors à la recherche des k meilleurs motifs (top_k) selon une mesure d'intérêt [9, 19]. Nous montrons, au travers de l'application à la découverte de fragments moléculaires toxicophores, comment les contraintes souples de seuil permettent d'extraire des motifs (très) pertinents qui n'auraient pas pu être découverts avec des seuils durs.

L'article est organisé comme suit. La section 2 présente le contexte et nos motivations. La section 3 décrit le cadre retenu de la relaxation disjonctive [14, 15] et montre comment les contraintes souples de seuil peuvent être transformées en contraintes dures équivalentes. La section 4 s'intéresse à la recherche des top_k motifs. La section 5 présente le cadre applicatif de la découverte de fragments moléculaires toxicophores en chémoinformatique. Les expérimentations menées (cf section 6) montrent l'apport des contraintes souples de seuil. La section 7 présente un état de l'art synthétique.

2 Contexte et motivations

2.1 Définitions

Soit $\mathcal{I}$ un ensemble de littéraux distincts appelés items. Un motif ensembliste d'items est un sous-ensemble non vide de $\mathcal{I}$. Ces motifs sont regroupés dans le langage $\mathcal{L}_\mathcal{I} = 2^\mathcal{I} \backslash \emptyset$. Un contexte transactionnel est un multi-ensemble de motifs de $\mathcal{L}_\mathcal{I}$. Chaque motif d'un contexte transactionnel constitue une entrée de la base de données et est appelé transaction. La table 1 présente un contexte transactionnel $\mathcal{T}$ dit "du panier du consommateur" où chaque transaction (panier) t_i rassemble les articles (items) achetés par le client i. Les 6 items sont notés $A, \ldots, F$. Enfin, à chaque article est associé un prix.

L'extraction de motifs sous contraintes a pour objectif d'extraire l'ensemble des motifs de $\mathcal{L}_\mathcal{I}$ présents dans $\mathcal{T}$ qui satisfont une requête (conjonction de contraintes).

La contrainte de seuil de fréquence permet d'extraire les motifs X_i dont le nombre d'occurrences dans

Trans.	Items					
t_1		B			E	F
t_2		B	C	D		
t_3	A				E	F
t_4	A	B	C	D	E	
t_5		B	C	D	E	
t_6		B	C	D	E	F
t_7	A	B	C	D	E	F

Items	A	B	C	D	E	F
Prix	30	40	10	40	70	55

TABLE 1 – Exemple de contexte transactionnel $\mathcal{T}$.

$\mathcal{T}$ dépasse un seuil minimal fixé par l'utilisateur : $\texttt{freq}(X_i) \geq min_{fr}$. D'autres mesures quantifient l'intérêt des motifs locaux recherchés. C'est le cas de la taille ou du taux de croissance :

Définition 1 (taux de croissance) *Soit $\mathcal{T}$ une base de données partitionnée en deux sous-ensembles $\mathcal{D}_1$ et $\mathcal{D}_2$. Le taux de croissance d'un motif X_i de $\mathcal{D}_2$ vers $\mathcal{D}_1$ est défini par :*

$$m_{gr}(X_i) = \frac{|\mathcal{D}_2| \times freq(X_i, \mathcal{D}_1)}{|\mathcal{D}_1| \times freq(X_i, \mathcal{D}_2)}$$

Définition 2 (Jumping Emerging Patterns) *Soit $\mathcal{T}$ une base de données partitionnée en deux sous-ensembles $\mathcal{D}_1$ et $\mathcal{D}_2$. X_i est un Jumping Emerging Pattern (JEP) ssi $m_{gr}(X_i) = +\infty$ (i.e. X_i ne figure pas dans $\mathcal{D}_2$).*

Par ailleurs, l'utilisateur est très souvent intéressé par la découverte de motifs plus riches que les motifs locaux et qui révèlent des caractéristiques et propriétés de tout l'ensemble des données étudiés. De tels motifs sont appelés motifs n-aires et leur extraction (basée-CSP) est décrite dans [10, 11]. L'approche décrite dans ce papier traite des motifs n-aires.

2.2 Motivations à l'aide d'un exemple

Exemple 2.1 *Considérons la requête ci-dessous, permettant d'extraire de la base $\mathcal{T}$, tous les motifs fréquents ($min_{fr} = 4$), de taille au moins 3 et dont la moyenne des prix des articles le composant est supérieure à 45 (mesure* `moyPrix`*) :*

$$\texttt{freq}(X_i) \geq 4 \wedge \texttt{taille}(X_i) \geq 3 \wedge \texttt{moyPrix}(X_i) \geq 45.$$

Par la suite, nous adoptons la notation X_i<v_1, v_2, v_3>, où X_i est un motif, et v_1, v_2, v_3 désignent les valeurs de ces trois mesures pour X_i.

Sur cet exemple, en considérant uniquement la contrainte de fréquence minimale, il y a 17 solutions.

Avec la conjonction des trois contraintes de la requête, il subsiste une unique solution : BDE <4, 3, 50>.

Considérons les 4 motifs suivants :
– $BEF < 3, 3, 55 >$ – $BCE < 4, 3, 40 >$
– $CDE < 4, 3, 40 >$ – $BCDE < 4, 4, 40 >$

Le motif BEF satisfait deux des trois contraintes de la requête Q, mais viole légèrement la contrainte de fréquence. Ce motif est clairement intéressant car il présente une dépense moyenne plus élevée que celle du motif BDE, qui satisfait la requête. Ainsi, en relâchant très légèrement le seuil de fréquence ($\texttt{freq}(X_i) \geq 3$), BEF serait solution.

De même, relâcher légèrement le seuil de la contrainte de prix moyen ($\texttt{moyPrix}(X_i) \geq 40$) permettrait d'extraire trois nouveaux motifs : CDE, BCE et $BCDE$. De plus, il est difficile d'affirmer que ces motifs sont nettement moins intéressants que le motif BDE compte-tenu de l'incertitude quant à la valeur fixée du seuil.

Ainsi, la rigidité du cadre de satisfaction fait qu'un motif potentiellement intéressant n'est pas retenu dès qu'une contrainte de seuil n'est pas vérifiée. Par ailleurs, il n'est pas raisonnable, dans des applications réelles, de considérer que les contraintes sont toutes d'une importance égale. D'où l'idée d'introduire une certaine souplesse, en relaxant les seuils, pour éviter une sélection trop dichotomique des motifs.

3 Mise en œuvre des contraintes souples de seuil

Nous présentons la problématique de la relaxation de contraintes ainsi que le cadre général de la relaxation disjonctive [14, 15]. Nous montrons comment les contraintes souples de seuil peuvent être transformées en des contraintes dures équivalentes et être directement résolues par un solveur de CSP.

3.1 Relaxation de contraintes en PPC

Relaxer une contrainte, c'est l'autoriser à ne pas être nécessairement satisfaite contre un coût. Pour cela, on distingue deux catégories de contraintes : les *contraintes d'intégrité* (ou dures) qui doivent être impérativement satisfaites, et les *contraintes de préférence* (ou souples) qui expriment des propriétés que l'on souhaiterait voir vérifiées par une solution. À chaque contrainte souple est associé un coût qui traduit son importance.

Le but de la relaxation n'est plus de satisfaire toutes les contraintes, mais de les satisfaire *au mieux*, i.e. satisfaire toutes les contraintes dures et minimiser la somme des coûts des contraintes souples insatisfaites. La relaxation se modélise sous forme d'un Problème d'Optimisation sous Contraintes (COP).

Une sémantique de violation μ pour une contrainte c permet de quantifier la violation de c lorsque c n'est pas satisfaite. À chaque instanciation $\mathcal{A}$ des variables de c, on associe la quantité de violation induite par $\mathcal{A}$ pour la contrainte c.

Définition 3 (sémantique de violation) *μ est une sémantique de violation pour la contrainte $c(X_1, ..., X_k)$ ssi μ est une fonction de $D_1 \times ... \times D_k$ vers $\mathbb{R}^+$ t.q. $\forall \mathcal{A} \in D_1 \times ... \times D_k$, $\mu(\mathcal{A}) = 0$ ssi $c(X_1, ..., X_k)$ est satisfaite.*

Pour une même contrainte, on peut définir plusieurs sémantiques de violation suivant les contextes d'utilisation. Ce sera le cas pour les contraintes de seuil étudiées à la section 3.3, pour lesquelles nous proposons deux sémantiques de violation différentes.

3.2 Cadre général de la relaxation disjonctive

La relaxation disjonctive [14, 15] s'intéresse au problème de satisfaction associé au COP : étant donné une quantité de violation maximale λ, existe-t-il au moins une instanciation de coût inférieur ou égal à λ ? La version relaxée de chaque contrainte est formulée sous forme d'une disjonction : soit la contrainte est satisfaite et le coût est nul, soit la contrainte est insatisfaite et le coût est précisé.

Définition 4 (relax. disjonctive d'une contrainte) *Soit c une contrainte, $\bar{c}$ sa négation et z la variable de coût associée. La relaxation disjonctive de c est la contrainte c' :*

$$c' = [c \wedge (z = 0)] \vee [\bar{c} \wedge (z > 0)]$$

Exemple 3.1 *Soit $\mathcal{X}$=$\{X_1, X_2\}$ de domaines D_1=D_2=$\{1, 2, 3\}$. Soit la contrainte X_1=X_2 avec comme sémantique de violation μ la distance entre X_1 et X_2, alors $z = |X_1 - X_2|$. La relaxation disjonctive de c est la suivante :*

$$c' = [X_1 = X_2 \wedge z = 0] \vee [X_1 \neq X_2 \wedge z = |X_1 - X_2|]$$

Soit C_s l'ensemble des contraintes souples et C_h l'ensemble des contraintes dures. À chaque contrainte $c_i \in C_s$, on associe une variable de coût z_i. Soit Z la variable représentant la violation totale (cumul des violations), alors $Z = \sum_{c_i \in C_s} z_i$. Soit λ la quantité maximale de violation que l'on s'autorise. Le problème de satisfaction associé se formule comme suit : Existe-t-il au moins une instanciation telle que $Z \leq \lambda$, i.e. $\sum_{c_i \in C_s} z_i \leq \lambda$.

Nous avons choisi le modèle de relaxation disjonctive car : (i) le fait que l'on puisse transformer une

contrainte souple en une, ou plusieurs contraintes dures équivalentes, permet de pouvoir traiter la relaxation avec des solveurs de CSP et de bénéficier des avancées faites dans ce domaine ; *(ii)* nous pouvons ainsi directement inclure cette forme de relaxation dans l'extracteur de motifs n-aires (basé CSP) que nous avions précédemment développé [10].

3.3 Sémantiques de violation pour les contraintes de seuil

Dans cette section, nous prenons comme exemple introductif la mesure de fréquence, puis nous traitons le cas d'une mesure quelconque.

3.3.1 Mesure de fréquence

Soit X_i un motif, α un seuil de fréquence et la contrainte $c = \texttt{freq}(X_i) \geq \alpha$. Une première sémantique de violation μ_1 consiste à mesurer l'écart absolu au seuil. Mais, pour pouvoir cumuler les violations des différentes contraintes de seuil, il est nécessaire de travailler avec des écarts relatifs. Une seconde sémantique de violation μ_2 consiste à associer, à chaque motif X_i, l'écart relatif de sa fréquence au seuil α :

$$\mu_2(X_i) = \begin{cases} 0 & si\ \texttt{freq}(X_i) \geq \alpha \\ \frac{\alpha - \texttt{freq}(X_i)}{\alpha} & sinon \end{cases}$$

3.3.2 Mesure m quelconque

Soit $\mathcal{I}$ un ensemble d'items et $\mathcal{T}$ un ensemble de transactions. Soit max_m la valeur maximale [1] pour la mesure m.

$$si \quad c = m(X_i) \geq \alpha \quad alors$$

$$\mu_2(X_i) = \begin{cases} 0 & si\ m(X_i) \geq \alpha \\ \frac{\alpha - m(X_i)}{\alpha} & sinon \end{cases}$$

$$si \quad c = m(X_i) \leq \alpha \quad alors$$

$$\mu_2(X_i) = \begin{cases} 0 & si\ m(X_i) \leq \alpha \\ \frac{m(X_i) - \alpha}{max_m - \alpha} & sinon \end{cases}$$

3.4 Transformation des contraintes souples de seuil

Cette section montre comment transformer des contraintes souples de seuil en contraintes dures équivalentes. Tout d'abord, nous modélisons le problème d'extraction des motifs n-aires sous forme de CSP. Puis, nous présentons la transformation en deux temps : un exemple introductif (fréquence) puis une mesure quelconque. Enfin, nous décrivons le CSP résultant associé à la relaxation dans le cadre disjonctif.

3.4.1 CSP initial

Soit $\mathcal{I}$ l'ensemble des items, $\mathcal{T}$ l'ensemble des transactions. Tout problème d'extraction de motifs n-aires peut se modéliser [10, 11] sous la forme d'un CSP $\mathcal{P}$=($\mathcal{X}, \mathcal{D}, \mathcal{C}$) tel que :

- $\mathcal{X} = \{X_1, ..., X_n\}$, chaque variable X_i représente un motif inconnu.
- $\mathcal{D} = \{D_{X_1}, ..., D_{X_n}\}$, chaque domaine D_{X_i} est l'intervalle ensembliste $[\emptyset\ ..\ \mathcal{I}]$.
- $\mathcal{C}$ est l'ensemble des contraintes que l'on peut partitionner en $\mathcal{C}_{ens} \cup \mathcal{C}_{num}$ où :
  - $\mathcal{C}_{ens}$ est un ensemble de contraintes ensemblistes. Exemples : $X_1 \subset X_2$, $I \in X_4$, ...
  - $\mathcal{C}_{num}$ est un ensemble de contraintes numériques sur les mesures. Exemples : $|\texttt{freq}(X_1) - \texttt{freq}(X_2)| \leq \alpha_1$, $\texttt{size}(X_4) < \texttt{size}(X_1) + 1$, ...

3.4.2 Mesure de fréquence

Soit X_i un motif, α un seuil de fréquence et la contrainte $c = \texttt{freq}(X_i) \geq \alpha$. Soit z la variable de coût associée. La relaxation disjonctive de c pour μ_2 est :

$$[(\texttt{freq}(X_i) \geq \alpha) \wedge z = 0] \vee [(\texttt{freq}(X_i) < \alpha) \wedge z = \frac{\alpha - \texttt{freq}(X_i)}{\alpha}]$$

Que l'on peut reformuler de manière équivalente par l'unique contrainte : $z = max(0, \frac{\alpha - \texttt{freq}(X_i)}{\alpha})$

3.4.3 Mesure m quelconque

En appliquant une transformation identique à celle de la mesure de fréquence, on obtient la reformulation, en contraintes dures équivalentes, des contraintes de seuil souples associées à m.

- La relaxation de $c = (m(X_i) \geq \alpha)$ est : $c' = [z = max(0, \frac{\alpha - m(X_i)}{\alpha})]$
- La relaxation de $c = (m(X_i) \leq \alpha)$ est : $c' = [z = max(0, \frac{m(X_i) - \alpha}{max_m - \alpha})]$

Ainsi, toute requête contenant une ou plusieurs contraintes souples de seuil c_i peut être transformée en une requête *équivalente* ne contenant que des contraintes dures : si c_i est une contrainte dure alors elle reste telle quelle ; si c_i est une contrainte souple de seuil alors elle est remplacée par sa transformée.

Enfin, soit λ la quantité maximale de violation autorisée. On définit la variable de coût $Z = \sum_{c_i} z_i$ représentant le cumul des violations, où z_i est la variable

1. Pour la fréquence, max_m=$|\mathcal{T}|$; pour la taille, max_m=$|\mathcal{I}|$.

de coût associée à chaque contrainte souple de seuil c_i. Enfin, on ajoute la contrainte $Z \leq \lambda$.

3.4.4 CSP issu de la transformation

Soit λ la quantité maximale de violation autorisée (λ compris entre 0 et 100%). Soit $\mathcal{P}'=(\mathcal{X}', \mathcal{D}', \mathcal{C}')$ le CSP obtenu par la relaxation disjonctive du CSP initial $\mathcal{P}=(\mathcal{X}, \mathcal{D}, \mathcal{C})$:

- $\mathcal{X}' = \mathcal{X} \bigcup_{1 \leq i \leq k} \{z_i\} \cup \{Z\}$,
- $\mathcal{D}' = \mathcal{D} \bigcup_{1 \leq i \leq k} \{D_{z_i}\} \cup \{D_Z\}$ avec D_{z_i}=[0..100] et D_Z=[0..λ] qui sont des intervalles d'entiers,
- $\mathcal{C}' = \mathcal{C}_{ens} \cup \mathcal{C}'_{num} \cup \{Z = \sum_{1 \leq i \leq k} z_i\}$ avec $\mathcal{C}'_{num} = \mathcal{C}_{hard} \cup \mathcal{C}_{disj}$ où :
  - $\mathcal{C}_{hard}$ est l'ensemble des contraintes numériques dures,
  - $\mathcal{C}_{disj}$ est l'ensemble des contraintes dures associées aux k contraintes souples de seuil.

4 Extraction des k-meilleurs motifs

En fouille de données, la recherche des k meilleurs motifs selon une mesure d'intérêt (top_k motifs) se révèle très utile pour trouver les motifs les plus significatifs au regard d'un critère choisi par l'utilisateur. Plusieurs méthodes adaptées à différentes mesures d'intérêt ont été proposées dans cette direction [9, 19].

Pour cela, on définit une mesure d'intérêt sur les motifs extraits, i.e. ceux satisfaisant la requête initiale, de façon à pouvoir les comparer.

4.1 Exemple introductif

Soit X_i un motif, μ une sémantique de violation et la contrainte de seuil $\texttt{freq}(X_i) \geq \alpha$. Une sémantique de violation ne permet pas de prendre en compte le degré de satisfaction d'une contrainte c. En effet, si un motif X_i satisfait une contrainte c alors $\mu(X_i) = 0$. Or un motif X_i, dont la fréquence est très grande par rapport au seuil α, sera considéré comme plus intéressant qu'un motif X_j dont la fréquence est légèrement supérieure à α.

4.2 Intérêt d'un motif pour une contrainte de seuil

Soit m une mesure, et max_m la valeur maximale pour cette mesure. Nous proposons la mesure d'intérêt $\theta_m : \{X_i \in \mathcal{L}_\mathcal{I}, \mu_2(X_i) \leq \lambda\} \rightarrow$ [-100..100] définie par :

$$si \quad c = m(X_i) \geq \alpha \quad alors$$

$$\theta_m(X_i) = \begin{cases} \frac{m(X_i)-\alpha}{max_m-\alpha} & si \ m(X_i) \geq \alpha \\ \\ \frac{m(X_i)-\alpha}{\alpha} & sinon \end{cases}$$

$$si \quad c = m(X_i) \leq \alpha \quad alors$$

$$\theta_m(X_i) = \begin{cases} \frac{\alpha-m(X_i)}{\alpha} & si \ m(X_i) \leq \alpha \\ \\ \frac{\alpha-m(X_i)}{max_m-\alpha} & sinon \end{cases}$$

4.3 Intérêt d'un motif pour une requête

Considérons à présent un ensemble de mesures $\mathcal{M}$ et une requête exprimée sous la forme d'une conjonction de contraintes de seuil de la forme $m(X_i) \geq \alpha_m$ (ou bien $\leq$). On définit l'intérêt d'un motif pour une requête (conjonction de contraintes) comme étant le cumul des intérêts des contraintes qui la composent :

$$\theta(X_i) = \sum_{m \in \mathcal{M}} \gamma_m \times \theta_m(X_i)$$

où γ_m est un coefficient traduisant l'importance de la mesure m. On peut alors extraire les top_k motifs, i.e. les k meilleurs motifs selon la mesure d'intérêt θ.

5 Découverte de fragments toxicophores

La toxicologie est la science étudiant les substances chimiques toxiques. Elle s'intéresse notamment à l'identification de fragments moléculaires spécifiques appelés toxicophores et considérés comme responsables des propriétés toxiques d'une substance chimique. Un objectif majeur est alors la découverte de tels fragments afin de mieux identifier les caractéristiques des molécules liées à la toxicité.

5.1 Fragments toxicophores

Un motif chimique émergent est une conjonction de fragments moléculaires qui apparaît fréquemment dans une classe de molécules et peu fréquemment dans une autre classe [1, 2]. L'émergence d'un motif chimique est mesurée par le taux de croissance entre les deux classes (cf définition 1). La combinaison des mesures d'*émergence* et de *fréquence* (afin d'assurer une certaine représentativité), s'avére précieuse pour la prédiction de la toxicité [16]. Par ailleurs, d'autres mesures issues des connaissances chimiques, comme *l'aromaticité* ou *la densité* d'une molécule, sont aussi des indicateurs connus de la toxicité. C'est pourquoi nous avons exploité ces différents types de mesures pour l'extraction des toxicophores (cf section 5.2).

Nous avons utilisé un jeu de données (base European Chemicals Bureau) préparé par le CERMN [2]. La toxicité est fondée sur l'indicateur quantitatif de toxicité

2. Centre d'Etudes et de Recherche sur le Médicament de Normandie, UPRES EA 4258 FR CNRS 3038, Université de Caen Basse-Normandie.

$CL50$ [3]. En fonction de la valeur de cet indicateur, les molécules sont réparties dans les trois catégories suivantes : $H400$ très toxique ($CL50 \leq 1$ mg/L), $H401$ toxique (1 mg/L $< CL50 \leq 10$ mg/L), et $H402$ nocif (10 mg/L $< CL50 \leq 100$ mg/L).

Dans cette étude, nous nous concentrons uniquement sur les classes $H400$ et $H402$. Le jeu de données utilisé contient 567 molécules, 372 de la classe $H400$ ($\mathcal{D}_1$) et 195 de la classe $H402$ ($\mathcal{D}_2$). Les molécules sont représentées en utilisant 129 sous-graphes connexes fréquents initialement extraits au seuil de fréquence de 10% [16].

5.2 Contraintes de seuil considérées

L'émergence permet de caractériser une molécule d'une classe (toxique) par rapport à une autre classe (non-toxique). Les motifs émergents traduisent l'hypothèse toxicophore (**H**1) : si une molécule possède dans sa structure les fragments moléculaires d'un motif émergent, alors elle possède des caractéristiques de toxicité et est donc particulièrement susceptible d'être toxique. L'émergence est mesurée par le taux de croissance m_{gr} (cf définition 1). Soit min_{gr} un seuil minimal pour le taux de croissance. On impose la contrainte souple de seuil : $m_{gr}(X_i) \geq min_{gr}$.

Fréquence. Les motifs de faible fréquence sont souvent dûs à des artefacts dans les données et constituent du bruit. Afin d'assurer une représentativité de l'information extraite, on impose la contrainte souple de seuil : $\texttt{freq}(X_i) \geq min_{fr}$, où min_{fr} est un seuil minimal pour la fréquence.

Aromaticité. L'intérêt de cette mesure est qu'elle véhicule une hypothèse toxicophore (**H**2) : plus la valeur d'aromaticité est forte, plus la molécule possédant ces fragments moléculaires tend à être toxique. L'aromaticité d'un motif est la moyenne de l'aromaticité de ses fragments moléculaires. Soit m_a la mesure d'aromaticité d'un motif. On impose la contrainte souple de seuil : $m_a(X_i) \geq min_a$, où min_a est un seuil minimal pour l'aromaticité.

Densité. Plus un sous-graphe codant une molécule est dense [4], plus son comportement chimique est fort. Un motif composé de fragments moléculaires denses renforce l'hypothèse toxicophore (**H**3). La densité d'un motif est la moyenne des densités de ses fragments. Soit m_d la mesure de densité et min_d un seuil minimal. On impose la contrainte souple de seuil : $m_d(X_i) \geq min_d$

3. Concentration nécessaire d'une substance pour causer la mort de 50% d'une population dans des conditions expérimentales précises.

4. La densité d'un sous-graphe est égale à $2e/v(v-1)$, où e est son nombre d'arêtes et v son nombre de sommets.

6 Expérimentations

6.1 Protocole expérimental

La requête soumise $\mathsf{q}(X_i)$ est la conjonction des quatre contraintes souples de seuil présentées à la section 5.2.

Pour l'aromaticité et la densité, les seuils ont été fixés à environ 2/3 de leur valeur maximale (min_a=60 et min_d=60). En effet, des seuils élevés sont en faveur des hypothèses toxicophores (**H**2) et (**H**3). Pour l'émergence et la fréquence, les seuils ont été fixés à environ 1/4 de leur valeur maximale (min_{gr}=5 et min_{fr}=90) de manière à obtenir un compromis entre la fréquence et le taux de croissance. Un tel choix nous permet de n'extraire que les motifs les plus fréquents ayant les meilleurs taux de croissance.

Nous avons retenu la sémantique de violation μ_2 (écart relatif) car les contraintes sont de nature hétérogène. Nous avons considéré trois valeurs différentes pour la quantité maximale de violation autorisée : $\lambda \in \{0, 20\%, 40\%\}$. Pour évaluer l'intérêt des motifs extraits, nous avons fixé γ_{gr}, γ_{fr} et γ_d à 1 et γ_a à 2. En effet, l'aromaticité constitue une connaissance chimique plus importante.

Pour évaluer la présence de toxicophores dans les motifs émergents de fragments moléculaires extraits, nous avons répertorié six fragments moléculaires ayant des propriétés écotoxicologiques connues, comme le benzène, le phénol, le chlorobenzène, les organophosphorés, l'aniline et le pyrrole.

Toutes nos expérimentations ont été réalisées sur un processeur Intel core i3 à 2,13 GHz ayant 4 Go de RAM. La mise en œuvre de notre approche a été réalisée en `Gecode` par extension de l'extracteur de motifs n-aires basé-CSP développé par M. Khiari [10].

6.2 Extraction des motifs émergents

Le tableau 2 indique les nombres de motifs émergents extraits contenant au moins un toxicophore dans son intégralité (colonnes notées **T**) ou des sous-fragments d'un toxicophore (colonnes notées **F**) parmi cinq des six fragments moléculaires répertoriés dans la base, ceci pour les trois valeurs de λ retenues (cf section 6.1).

Cette répartition est donnée pour le nombre total de solutions trouvées (col. 2-7) mais aussi pour les top_{25} (col 8-13) et les top_{50} (col 14-19). Comme les deux catégories **T** et **F** ne sont pas disjointes, la somme de leurs nombres de motifs est toujours supérieure ou

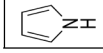
	Total						*top* − 25						*top* − 50					
λ	0		20		40		0		20		40		0		20		40	
# Solutions	**7650**		**402204**		**4289335**		**28**		**37**		**57**		**55**		**64**		**85**	
	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Benzène c1ccccc1 	1912	7573	183881	396749	1565883	4210482	0	25	**2**	25	**6**	24	7	50	**7**	50	**8**	49
Phénol c1(ccccc1)O 	900	4519	93632	217195	556890	3234279	2	9	**6**	3	**2**	0	4	18	**9**	12	**5**	8
Chlorobenzène Clc1ccccc1 	0	3041	74182	184502	253429	509281	0	14	**2**	14	**2**	1	0	28	**7**	22	**2**	15
Aniline c1(ccccc1)N 																		
Pyrrole c1cncc1 						1						1						1

TABLE 2 – Répartition des motifs émergents en fonction des toxicophores connus.

égale au nombre total de solutions (# solutions). Le temps nécessaire pour extraire l'ensemble de toutes les solutions est de 21 s. pour (λ=0), 9 min. pour (λ=20) et 6h15 min. pour (λ=40).

Comme le montrent les résultats du tableau 2, 45%[5](resp. 36.5%) des solutions extraites avec λ=20 (resp. 40) contiennent du benzène (fragment de type **T**), contre environ 25% pour λ=0. Les seuils souples permettent ainsi de mieux retrouver ce toxicophore (gain moyen d'environ 16%). Pour les fragments de type **F**, la proportion de solutions extraites contenant des sous-fragments du benzène ($\{cc, ccc, cccc, ccccc\}$) est quasi identique dans le cas dur et le cas souple (environ 98%). Cette tendance se confirme sur le phénol, où 23% (resp. 13%) des solutions extraites avec λ=20% (resp. 40%) contiennent un tel fragment, contre 11% pour λ=0. De nouveau, les seuils souples permettent de mieux retrouver ce toxicophore (gain moyen d'environ 7%).

Pour le chlorobenzène (avec λ=0), seuls les motifs contenant des fragments moléculaires de type **F** sont extraits : $\{Clc(c)cc, Clc(c)ccc, Clc(c)cccc, Clc(cc)ccc, Clccc\ldots\}$. Les seuils souples permettent de retrouver en moyenne 19% de toxicophones contenant du chlorobenzène (i.e, fragment de type **T**). De plus, pour le pyrrole, les seuils souples permettent de détecter un nouveau motif contenant le sous-fragment nc. Sans les seuils souples, ce motif serait difficile à extraire car associé à un sous-fragment moléculaire ayant une valeur de fréquence relativement faible.

Les motifs contenant de l'aniline ne sont pas détectés en raison de leur faible densité (33). En effet, pour λ=40, la valeur minimale autorisée est de 60×0.60=36. Une augmentation très légère (λ=45), permettrait d'extraire ces motifs. Enfin, les motifs organo-phosphorés sont caractérisés par une très forte émergence ($+\infty$). Ce sont des JEPs (cf définition 2) : ils ne figurent pas dans le tableau 2 et sont traités à part dans la section 6.4).

6.3 Extraction des top_k motifs

Les résultats du tableau 2 montrent que sur les top_{25} (resp. top_{50}) motifs extraits avec λ=0, seuls 2 (resp. 4) motifs contiennent du phénol. Par ailleurs, les top_k motifs extraits sont constitués uniquement de sous-fragments de benzène ou de chlorobenzène.

Le tableau 3 donne les top_{25} motifs émergents extraits avec λ=20. Les lignes jaunes indiquent les motifs extraits avec λ=0 et contenant du phénol dans son intégralité, alors que les lignes grisées correspondent aux nouveaux motifs obtenus grâce aux contraintes souples de seuil (les seuils violés sont sur-lignés en noir).

Les seuil souples permettent de retrouver 4 nouveaux motifs contenant du phénol sur les top_{25} extraits (lignes 17 − 20), ce qui représente un ratio de 3 (λ=20 permet de détecter 3 fois plus de meilleures solutions comparé à λ=0). Notons que 2 de ces motifs contiennent également du benzène (lignes 18 et 20). Par ailleurs, ces motifs, qui violent très légèrement la contrainte de densité, sont caractérisés par une très forte aromaticité et une forte émergence, ce qui renforce nos hypothèses toxicophores sur l'émergence (**H**1) et l'aromaticité (**H**2). De plus, λ=20 permet d'extraire 2 nouveaux motifs contenant du chlorobenzène (lignes 2 et 4) et un motif contenant le fragment $Clc(cc)ccc$ (ligne 10). Ces motifs sont aussi intéressants, car ils renforcent les hypothèses toxicophores précédemment émises.

5. Ratio entre le nombre de solutions contenant un toxicophore et le nombre total de solutions.

N	Intérêt	Motif	Émergence	Fréquence	Aromaticité	Densité	SMILES[6]	Condensée
1	193	24 35 69	7	101	95	66	cc ccc c1(ccccc1)O	c1(ccccc1)O
2	191	13 24 35	8	89	95	66	Clc1ccccc1 cc ccc	Clc1ccccc1
3	189	24 35 47 69	7	101	96	62	cc ccc cccc c1(ccccc1)O	c1(ccccc1)O
4	187	13 24 35 47	8	89	96	62	Clc1ccccc1 cc ccc cccc	Clc1ccccc1
5	185	12 24 35 47	8	90	96	61	Clc(c)cccc cc ccc cccc	Clc(c)cccc
6	185	14 24 35 47	8	90	96	61	Clccccc cc ccc cccc	Clccccc
7	185	24 35 47 68	6	103	96	61	cc ccc cccc ccccc(c)O	ccccc(c)O
8	185	24 35 47 80	6	103	96	61	cc ccc cccc ccc(ccc)O	ccc(ccc)O
9	185	24 35 38	5	118	90	72	cc ccc cccO	cccO
10	184	24 35 47 78	8	89	96	61	cc ccc cccc Clc(cc)ccc	Clc(cc)ccc
11	184	6 24 35	9	93	90	72	Clc(c)c cc ccc	Clc(c)c
12	184	8 24 35 47	9	93	94	64	Clc(c)cc cc ccc cccc	Clc(c)cc
13	184	8 24 35	9	93	92	68	Clc(c)cc cc ccc	Clc(c)cc
14	184	7 24 35	9	94	90	72	Clccc cc ccc	Clccc
15	184	9 24 35 47	9	94	94	64	Clcccc cc ccc cccc	Clcccc
16	184	9 24 35	9	94	92	68	Clcccc cc ccc	Clcccc
17	183	24 35 47 59 69	7	101	97	57	cc ccc cccc ccccc c1(ccccc1)O	c1(ccccc1)O
18	183	24 35 47 69 77	7	101	97	57	cc ccc cccc c1(ccccc1)O c1ccccc1	c1(ccccc1)O
19	183	24 35 59 69	7	101	96	59	cc ccc ccccc c1(ccccc1)O	c1(ccccc1)O
20	183	24 35 69 77	7	101	96	59	cc ccc c1(ccccc1)O c1ccccc1	c1(ccccc1)O
21	183	12 24 35	8	90	94	64	Clc(c)cccc cc ccc	Clc(c)cccc
22	183	14 24 35	8	90	94	64	Clccccc cc ccc	Clccccc
23	183	11 24 35 47	9	92	95	62	Clcccc cc ccc cccc	Clcccc
24	183	11 24 35	9	92	93	66	Clcccc cc ccc	Clcccc
25	183	10 24 35 47	9	93	95	62	Clc(c)ccc cc ccc cccc	Clc(c)ccc

[6] http ://www.daylight.com/dayhtml/doc/theory/theory.smiles.html

TABLE 3 – top_{25} motifs émergents extraits avec λ=20.

Le tableau 4 montre les top_{25} motifs émergents extraits avec λ=40. Comme précédemment, les contraintes souples de seuil permettent de retrouver 6 nouveaux motifs contenant du benzène (cf. lignes 7, 9, 14, 16, 19 et 20). Ces motifs, qui violent très légèrement le seuil d'émergence, sont très fortement aromatiques et relativement denses, ce qui renforce de nouveau l'hypothèse toxicophore liée à la densité (**H3**). Un nouveau motif particulièrement intéressant pour les chimistes est obtenu : $\{nc\}$. Ce motif, incluant un sous-fragment du pyrrole, est réputé dangereux pour l'environnement car il est très toxique pour les espèces aquatiques.

Par ailleurs, si l'on considère les top_{50} motifs extraits, les seuils souples λ=20 (resp. 40) permettent de détecter 2.25 (resp. 1.25) fois plus de meilleures solutions contenant du phénol. De plus, λ=40 permet d'extraire 8 (resp. 2) nouveaux motifs contenant du benzène (resp. chlorobenzène). Ainsi, l'ensemble de ces résultats confirme l'intérêt des seuils souples pour l'extraction de motifs toxicophores.

6.4 Extraction des Jumping Emerging Patterns

Cette seconde série d'expérimentations évalue le caractère de toxicité porté par les fragments moléculaires présents uniquement dans les molécules très toxiques (classe $H400$), autrement dit les Jumping Emerging Patterns (cf définition 2). Le tableau 5, présente les top_{25} JEPs extraits pour différentes valeurs de λ. On peut dresser les remarques suivantes :

(i) Sans les seuils souples, aucun JEP n'a pu être extrait.

(ii) Avec λ=50 (resp. 60), nous obtenons 3 (resp. 457) JEPs. En effet, ces motifs sont peu fréquents, il est donc nécessaire d'avoir un seuil de violation relativement élevé.

(iii) Tous les motifs contenant des sous-fragments organo-phosphorés ont un taux de croissance infini. Ce composant est une généralisation de plusieurs Jumping Emerging Fragments et peut être vu comme unestructure commune maximale de ces fragments.

(iv) Parmi les top_{25} motifs extraits avec λ=60, les motifs les plus intéressants sont ceux contenant un *cycle benzénique* ($c1ccccc1$). En effet, le benzène est un fragment moléculaire très aromatique. Pour λ=50, seuls les motifs contenant des sous-fragments du benzène sont extraits. Ces motifs sont malgré tout moins pertinents d'un point de vue chimique.

A nouveau, ces résultats montrent la pertinence et l'apport des contraintes souples de seuil pour mettre en évidence les structures chimiques les plus parlantes, comme par exemple les cycles benzéniques par rapport à leurs sous-fragments.

7 Travaux relatifs

En fouille de données, la relaxation a été utilisée pour obtenir des contraintes relaxées ayant des propriétés de monotonie dans le but de réutiliser les algorithmes usuels de filtrage. Ainsi, les contraintes basées sur des expressions régulières sont relaxées en des contraintes anti-monotones pour extraire des séquences [7]. Dans [18], les auteurs proposent de générer automatiquement une relaxation monotone ou anti-monotone d'une contrainte.

N	intérêt	motif	émergence	fréquence	aromaticité	densité	SMILES	Condensée
1	301	24	3	289	100	100	cc	cc
2	275	15	7	65	100	100	nc	nc
3	258	24 35	3	288	100	83	cc ccc	ccc
4	237	24 47	3	281	100	75	cc cccc	cccc
5	230	24 35 47	3	281	100	72	cc ccc cccc	cccc
6	224	24 59	3	279	100	70	cc ccccc	ccccc
7	223	24 77	3	274	100	70	cc c1ccccc1	c1ccccc1
8	219	24 35 59	3	279	100	68	cc ccc cccc	cccc
9	218	24 35 77	3	274	100	68	cc ccc c1ccccc1	c1ccccc1
10	216	35	3	288	100	65	ccc	ccc
11	213	24 35 76	3	274	100	66	cc ccc cccccc	cccccc
12	213	24 76	3	274	100	66	cc ccccccc	cccccc
13	209	24 35 47 59	3	279	100	64	cc ccc cccc ccccc	ccccc
14	208	24 35 47 77	3	274	100	64	cc ccc cccc c1ccccc1	c1ccccc1
15	206	24 47 59	3	279	100	63	cc cccc ccccc	ccccc
16	205	24 47 77	3	274	100	63	cc cccc c1ccccc1	c1ccccc1
17	203	24 35 47 76	3	274	100	62	cc ccc cccc cccccc	cccccc
18	200	24 47 76	3	274	100	61	cc cccc ccccc	ccccc
19	200	24 35 59 77	3	274	100	61	cc ccc cccc c1ccccc1	c1ccccc1
20	198	24 59 77	3	274	100	60	cc ccccc c1ccccc1	c1ccccc1
21	193	24 35 69	7	101	95	66	cc ccc c1(ccccc1)O	c1(ccccc1)O
22	191	13 24 35	8	89	95	66	Clc1ccccc1 cc ccc	Clc1ccccc1
23	189	24 35 47 69	7	101	96	62	cc ccc cccc c1(ccccc1)O	c1(ccccc1)O
24	187	13 24 35 47	8	89	96	62	Clc1ccccc1 cc ccc cccc	Clc1ccccc1
25	185	12 24 35 47	8	90	96	61	Clc(c)cccc cc ccc cccc	Clc(c)cccc

TABLE 4 – top_{25} motifs émergents extraits avec λ=40.

Pour les motifs locaux, [4, 5] ont proposé un cadre formel pour l'expression de préférences entre solutions. Chaque contrainte possède sa propre mesure d'intérêt et l'intérêt d'une requête est une agrégation des intérêts des contraintes qui la composent. Étant donnée une requête, il s'agit de trouver tous les motifs (locaux) dont la mesure d'intérêt satisfait un seuil.

Mais, cette approche repose sur une hypothèse très forte : l'intérêt d'une requête satisfait un seuil, si et seulement si, l'intérêt de *chaque* contrainte satisfait ce même seuil [4, 5]. Si on agrège les coûts par l'opérateur *min* (cas des *fuzzy sets*), alors on a bien l'équivalence. Par contre, dans le cadre additif que nous traitons (et aussi dans le cadre probabiliste), ce n'est plus le cas. C'est pourquoi les auteurs ont besoin de faire d'une étape supplémentaire (post-traitement) pour ne garder que les solutions satisfaisant le seuil.

Ainsi, à la différence de [4, 5], l'approche que nous proposons (i) préserve l'équivalence et ne nécessite donc aucun post-traitement, et (ii) elle s'applique aux motifs n-aires, et donc aux motifs locaux (unaires).

8 Conclusion

Nous avons proposé un cadre général permettant de mettre en œuvre les contraintes souples de seuil dans un extracteur de motifs (basé-CSP) en utilisant les travaux en PPC sur la relaxation de contraintes et les préférences entre solutions. Puis, nous avons montré l'apport et la faisabilité de notre approche au travers d'une application en chémoinformatique portant sur la découverte de fragments moléculaires toxicophores. Les premiers résultats expérimentaux montrent l'apport des contraintes souples de seuil pour mettre en évidence les structures chimiques les plus parlantes, comme par exemple les top_k motifs ou les JEPs.

Comme travaux futurs, nous souhaitons étudier l'apport des contraintes souples de seuil pour la résolution des problèmes de clustering sous contraintes [3], et pour le calcul des *skylines* qui constituent des points d'intérêt, non dominés par d'autres points, dans un espace [6].

Références

[1] J. Bajorath. Simulation of Sequential Screening Experiments Using Emerging Chemical Patterns. *Medicinal Chemistry*, 4 :80–90, 2008.

[2] J. Bajorath and J. Auer. Emerging chemical patterns : A new methodology for molecular classification and compound selection. *J. of Chemical Information and Modeling*, 46 :2502–2514, 2006.

[3] S. Basu, I. Davidson, and Kiri L. Wagstaff. *Constrained Clustering : Advances in Algorithms, Theory, and Applications*. Chapman & Hall, 2008.

[4] Stefano Bistarelli and Francesco Bonchi. Interestingness is not a dichotomy : Introducing softness in constrained pattern mining. In *Knowledge Discovery in Databases (PKDD'05)*, volume 3721 of *LNCS*, pages 22–33. Springer, 2005.

[5] Stefano Bistarelli and Francesco Bonchi. Soft constraint based pattern mining. *Data Knowl. Eng.*, 62(1) :118–137, 2007.

[6] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE 2001*, pages 421–430, 2001.

[7] M. N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT : Sequential pattern mining with regular expres-

N	intérêt	motif	émergence	fréquence	aromaticité	densité	SMILES	Condensée
$\lambda = 50$							# Solutions=3	
1	253	24 35 87	∞	47	66	88	cc ccc OP	ccc OP
2	222	24 35 90	∞	45	66	77	cc ccc OPO	ccc OPO
3	222	24 35 105	∞	45	66	77	cc ccc COP	ccc COP
$\lambda = 60$							# Solutions=457	
1	174	24 35 47 59 77 87	∞	40	83	66	cc ccc cccc ccccc c1ccccc1 OP	c1ccccc1 OP
2	174	24 35 47 59 87	∞	42	80	71	cc ccc cccc ccccc OP	ccccc OP
3	172	24 35 47 77 87	∞	40	80	71	cc ccc cccc c1ccccc1 OP	c1ccccc1 OP
4	171	24 35 47 59 76 77 87	∞	40	85	61	cc ccc cccc ccccc cccccc c1ccccc1 OP	c1ccccc1 OP
5	169	24 35 47 59 76 87	∞	40	83	64	cc ccc cccc ccccc cccccc OP	cccccc OP
6	169	24 35 47 76 77 87	∞	40	83	64	cc ccc cccc cccccc c1ccccc1 OP	c1ccccc1 OP
7	168	24 35 47 87	∞	42	75	79	cc ccc cccc OP	cccc OP
8	167	24 35 47 76 87	∞	40	80	69	cc ccc cccc cccccc OP	cccccc OP
9	167	24 35 59 77 87	∞	40	80	69	cc ccc ccccc c1ccccc1 OP	c1ccccc1 OP
10	166	24 35 59 76 77 87	∞	40	83	63	cc ccc ccccc cccccc c1ccccc1 OP	c1ccccc1 OP
11	162	24 35 59 76 87	∞	40	80	67	cc ccc ccccc cccccc OP	cccccc OP
12	162	24 35 76 77 87	∞	40	80	67	cc ccc cccccc c1ccccc1 OP	c1ccccc1 OP
13	161	24 35 59 87	∞	42	75	76	cc ccc ccccc OP	ccccc OP
14	160	24 47 59 77 87	∞	40	80	66	cc cccc ccccc c1ccccc1 OP	c1ccccc1 OP
15	159	24 35 77 87	∞	40	83	60	cc ccc c1ccccc1	c1ccccc1 OP
16	159	24 47 59 76 77 87	∞	40	75	76	cc cccc ccccc cccccc c1ccccc1 OP	c1ccccc1 OP
17	157	24 35 59 76 77 87	∞	38	83	60	cc ccc ccccc cccccc c1ccccc1 OP	c1ccccc1 OP
18	157	24 35 47 59 77 90	∞	38	83	60	cc ccc ccccc cccccc c1ccccc1 OPO	c1ccccc1 OPO
19	156	24 35 47 76 77 105	∞	38	83	59	cc ccc cccc cccccc c1ccccc1 COP	c1ccccc1 COP
20	156	24 35 47 59 76 105	∞	38	83	59	cc ccc cccc ccccc cccccc COP	cccccc COP
21	156	24 35 47 76 77 90	∞	38	83	59	cc ccc cccc cccccc c1ccccc1 OPO	c1ccccc1 OPO
22	156	24 35 47 59 76 90	∞	38	83	59	cc ccc cccc ccccc cccccc OPO	cccccc OPO
23	155	24 47 76 77 87	∞	40	80	64	cc cccc cccccc c1ccccc1 OP	c1ccccc1 OP
24	155	24 47 59 76 87	∞	40	80	64	cc cccc ccccc OP	ccccc OP
25	155	24 35 47 59 105	∞	40	80	64	cc ccc cccc ccccc COP	ccccc COP
26	155	24 35 47 59 90	∞	40	80	64	cc ccc cccc ccccc OPO	ccccc OPO

TABLE 5 – top_{25} Jumping Emerging Patterns extraits (λ=50 et λ=60).

sion constraints. *In The VLDB Journal*, pages 223–234, 1999.

[8] David J. Hand. Pattern detection and discovery. In *Pattern Detection and Discovery*, pages 1–12, 2002.

[9] Y. Ke, J. Cheng, and J. Xu Yu. Top-k correlative graph mining. In *SDM*, pages 1038–1049, 2009.

[10] M. Khiari, P. Boizumault, and B. Crémilleux. Constraint programming for mining n-ary patterns. In *CP'10*, volume 6308 of *LNCS*, pages 552–567. Springer, 2010.

[11] M. Khiari, P. Boizumault, and B. Crémilleux. Combining CSP and constraint-based mining for pattern discovery. In *Advances in Knowledge Discovery and Management 2 (Post-EGC'10 Selected Papers)*, Studies in Computational Intelligence, Vol. 398, pages 85–104. Springer, March 2012.

[12] K. Morik, J.F. Boulicaut, and A. Siebes, editors. *Local Pattern Detection, International Seminar, Dagstuhl Castle, Germany, April 12-16, 2004, Revised Selected Papers*, volume 3539 of *LNCS*. Springer, 2005.

[13] S. Nijssen and T. Guns. Integrating constraint programming and itemset mining. In *European Conference, ECML PKDD 2010*, volume 6322 of *LNCS*, pages 467–482. Springer, 2010.

[14] T. Petit, J.-C. Régin, and C. Bessière. Specific filtering algorithms for over-constrained problems. In *CP'01*, volume 2239 of *LNCS*, pages 451–463. Springer, 2001.

[15] T. Petit, J.-C. Régin, C. Bessière, and J.-P. Puget. An original constraint based approach for solving over constrained problems. In *CP'2000*, volume 1894 of *LNCS*, pages 543–548. Springer, 2000.

[16] G. Poezevara, B. Cuissart, B. Crémilleux, S. Lozano, M.-P. Halm-Lemeille, E. Lescot-Fontaine, A. Lepailleur, R. Bisell-Siders, S. Rault, and R. Bureau. Introduction of Jumping Fragments in Combination with QSARs for the Assessment of Classification in Ecotoxicology. *Journal of Chemical Information and Modeling(JCIM)*, pages 1330–1339, 2010.

[17] L. De Raedt, T. Guns, and S. Nijssen. Constraint programming for itemset mining. In *KDD'08*, pages 204–212. ACM, 2008.

[18] A. Soulet and B. Crémilleux. Optimizing constraint-based mining by automatically relaxing constraints. *In Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, pages 777–780, 2005.

[19] Jianyong Wang, Jiawei Han, Ying Lu, and Petre Tzvetkov. Tfp : An efficient algorithm for mining top-k frequent closed itemsets. *IEEE Trans. Knowl. Data Eng.*, 17(5) :652–664, 2005.

CoFiADe
Constraints Filtering for Aiding Design

Élise Vareilles, Paul Gaborit,
Michel Aldanondo, Sabine Carbonnel, Laurent Steffan

Université de Toulouse - École des Mines d'Albi-Carmaux
Route de Teillet - Campus Jarlard
81000 Albi Cedex 09
{prenom.nom}@mines-albi.fr

Résumé

Les systèmes interactifs d'aide à la conception et à la configuration sont destinés à faciliter les prises de décision en se basant sur des connaissances pertinentes et de différentes natures. En exploitant les *CSP*, ils permettent d'accompagner efficacement cette prise de décision, tout en garantissant leur interactivité, par l'utilisation d'algorithmes de filtrage. *CoFiADe* est un outil interactif dédié à l'aide à la décision en conception ou configuration. Il permet d'une part, aux experts de formaliser les connaissances du problème sous la forme d'un *CSP* via un langage de définition spécifique, et d'autre part, aux utilisateurs de piloter une interface Web, leur permettant via une vision arborescente du modèle, de réduire progressivement le domaines des variables pour converger vers une solution.

Pour cela, *CoFiADe* supporte la définition de variables de différentes natures (discrètes et continues) ainsi que des contraintes de différentes natures (compatibilité, activation) et exploite plusieurs méthodes de filtrage. *CoFiADe* est développé et maintenu en Perl à l'école des mines d'Albi-Carmaux et a d'ores et déjà servi de support à plusieurs applications industrielles. Dans un premier temps, nous exposerons les spécificités (mixité, multi-intervalles, activation) de *CoFiADe*, puis nous illustrerons ses capacités sur un exemple simple d'aide à la décision tiré d'une application industrielle et conclurons sur les améliorations à apporter afin de promouvoir son utilisation.

Abstract

Interactive aiding design or configuration tools support and help decisions making by using relevant knowledge of various origins. Product design and configuration can be efficiently modeled and aided when considered as a *CSP*. Their interactivity can then be guaranteed through the use of filtering algorithms. *CoFiADe* is an interactive aiding design or configuration tool. It makes it possible, on the first hand for experts to model their knowledge as a *CSP* via a specific language and, on the other hand, for users to reach a solution by progressively reducing the domain of the variables through a Web interface.

In order to achieve this goal, *CoFiADe* supports the definition of various kinds of variables (discrete, continuous), as well as various types of constraints (compatibility, activation) and it uses various filtering algorithms. *CoFiADe* has been developed in Perl and is kept up to date by people at école des mines d'Albi-Carmaux (France). Note that *CoFiADe* has already been used in several industrial projects in order to help users make better decisions. In the second section, the general features of *CoFiADe* (mixed problem, multi-interval, activity) are presented. In the third section, its abilities in term of aiding design are illustrated with a simple but real example. In the last section, we focus on the improvements that can be achieved in order to improve *CoFiADe* and promote its utilization.

1 Introduction

Les systèmes interactifs d'aide à la conception et à la configuration sont destinés à faciliter les prises de décision en se basant sur des connaissances pertinentes [1] et de différentes natures [2]. En exploitant les *CSP*, ils permettent d'accompagner efficacement cette prise de décision [3] [4], tout en garantissant leur interactivité, par l'utilisation d'algorithmes de filtrage. De nombreux travaux en aide à la conception dans différents domaines ont utilisé les approches

par contraintes : en conception d'appareils à pression [5], en conception aéronautique [6], en conception de produits à forte diversité [7], en conception de batteries électriques [8] ou en conception de matériaux multi-couches [9]. À travers cet échantillon d'applications, nous pouvons percevoir la diversité des connaissances à modéliser afin d'aider à la décision en conception ou configuration : connaissances sur les propriétés et fonctionnalités du produit (caractéristiques du produit, discrètes ou numériques comme la consommation ou le poids), sur les composants du produit (liste de composants à intégrer, majoritairement discret) et sur les options du produits (composants à concevoir ou configurer sous certaines conditions) [10].

CoFiADe est un outil interactif dédié à l'aide à la décision en conception et à la configuration. Il permet d'une part, aux experts de formaliser les connaissances du problème sous forme d'un *CSP* contenant des variables de différents types (discrètes et numériques) et des contraintes de différentes natures (compatibilité et activation) et de différents types (tables de compatibilité et expressions mathématiques). Pour cela, *CoFiADe* possède son propre langage de définition de modèle *CSP*. Il permet d'autre part, aux utilisateurs de piloter une interface Web, `http://cofiade.enstimac.fr/cgi-bin/cofiade.pl`, leur permettant, via une vision arborescente du modèle, de réduire progressivement le domaines des variables pour converger vers une solution. *CoFiADe*, écrit en Perl, est développé et maintenu à l'école des mines d'Albi-Carmaux. Il est issu de recherches menées au sein de notre laboratoire [2], [1], [10] et [11]. Il a servi de support au projet européen *VHT*[1], au projet ANR *ATLAS*[2] et au projet FUI *Helimaintenance*[3].

Dans le cadre de cette communication, nous exposons dans un premier temps, les spécificités de *CoFiADe* (mixité, multi-intervalles, activation) et préciserons les méthodes de filtrage utilisées. Dans un deuxième temps, nous illustrons ses capacités sur un exemple simple d'aide à la décision tiré d'une application industrielle [12]. Puis, nous concluons sur les suites à donner afin d'améliorer et promouvoir son utilisation.

2 Spécificités de *CoFiADe*

CoFiADe présente certaines spécificités que n'ont pas les autres outils basés sur les *CSP*, tels qu'*Ilog CP* [13] (`http://www.ilog.com/products/cp/`), *Choco* [14] (`http://www.emn.fr/z-info/choco-solver/`) ou *ECLiPSe* [15] (`http://87.230.22.228/`). Dans cette section, nous abordons ses spécificités du point de vue du modèle (variables et contraintes de différents types et natures) et du point de vue traitement (intégration de plusieurs méthodes de filtrage de la littérature).

2.1 Différents types de variables

CoFiADe, comme *Ilog CP*, *Choco* et *ECLiPSe*, permet de modéliser plusieurs types de variables mais sans inclusion de librairies spécifiques :

- les variables discrètes dont les domaines de définition sont finis ou dénombrables. Ceux-ci peuvent contenir un ensemble d'éléments soit de type symbolique, soit de type numérique (intervalles de valeurs entières, liste de valeurs entières ou réelles),
- les variables continues dont les domaines de définition sont indénombrables. Ceux-ci contiennent des éléments de type numérique (intervalles de valeurs réelles).

L'une des spécificités de *CoFiADe* est de pouvoir associer aux variables des domaines continus multi-intervalles [2]. Dans ce cas, les domaines ne sont pas constitués d'un unique intervalle de valeurs mais de plusieurs intervalles de valeurs disjoints.

2.2 Natures et types de contraintes

CoFiADe, comme *Ilog CP*, *Choco* et *ECLiPSe*, permet de modéliser plusieurs natures et types de contraintes. Nous distinguons deux natures de contraintes, celles de compatibilité et d'activation et deux types de contraintes, les tables de compatibilité et les expressions mathématiques. La figure **??** synthétise les notions de nature et type de contraintes gérées par *CoFiADe* [2].

Contraintes de différentes natures : Les contraintes de compatibilité permettent de restreindre l'espace de recherche en délimitant les combinaisons de valeurs que les variables peuvent prendre simultanément [16]. Elles peuvent être exprimées en *extension* ou en *intension*. Les contraintes de compatibilité exprimées en extension sont représentées par des listes de n-uplets indiquant quelles sont les valeurs compatibles entre elles. Les contraintes de compatibilité exprimées en intension concentrent les informations pour plusieurs raisons. Tout d'abord, il peut être plus facile d'écrire la contrainte en intension [17] (pour éviter de lister toutes les possibilités autorisées) ou parce qu'il peut être tout à fait impossible de lister de manière complète les combinaisons de valeurs [18].

Les contraintes d'activation permettent d'activer (inclure) ou de supprimer (exclure) des variables du

1. Virtual Heat Treatment, projet n° G1RD-CT-2002-00835
2. Aides et assisTances pour la conception, la conduite et leur coupLage par les connAissanceS, projet n° ANR-07-TLOG-002
3. projet pôle de compétitivité Aerospace Valley financé par FUI-DGE

problème courant [19]. C'est la détection d'événements singuliers qui déclenche l'ajout ou l'exclusion d'un ensemble d'éléments. Les contraintes d'activation sont définies par deux parties : une prémisse (contrainte de compatibilité) et un conséquent (ajout ou retrait d'éléments) liées par un opérateur d'implication $\rightarrow$. Si la prémisse est vraie, il y a application du conséquent. Sinon, le problème reste inchangé.

CoFiADe permet de définir et traiter dans un même problème, des contraintes de compatibilité décrites en extension ou en intension. En ce qui concerne les contraintes d'activation, il intègre une extension du concept d'activation aux contraintes et aux sous-problèmes, tels qu'introduit par [3]. Les contraintes d'activation manipulées par *CoFiADe* autorisent explicitement l'ajout d'ensembles de variables, de contraintes et de sous-problèmes selon le modèle des $RequireVariable$ de Mittal [19]. Les variables, les contraintes et les groupes possèdent donc un état *actif* ou *inactif* indiquant leur participation au problème courant. Dans *CoFiADe*, de manière identique aux *DCSP*, les variables *inactives* ne sont pas considérées dans le problème courant. Par contre, les contraintes *actives* sont toujours prises en compte et filtrées quelque soit l'état (*actif* ou *inactif*) des variables liées par les contraintes alors que les contraintes *inactives* ne sont jamais prises en compte et filtrées quelque soit l'état (*actif* ou *inactif*) des variables liées par celle-ci.

Contraintes de différents types : Les tables de compatibilité permettent de représenter sous forme tabulaire des listes de n-uplets de valeurs autorisées. Chaque combinaison de n-uplet est mise sous forme de tuple autorisé. La combinatoire exprimée sous forme de table de compatibilité correspond à un sous-ensemble du produit cartésien des domaines des variables participant à la contrainte. Les tables de compatibilité peuvent être soit discrètes, soit continues, soit mixtes si elles lient à la fois des variables discrètes et continues [12].

Les expressions mathématiques sont essentielles pour calculer des paramètres de conception ou pour représenter des lois physiques et des résultats d'expérimentations. Certaines relations peuvent se mettre sous la forme d'une fonction numérique de $\mathbb{R}^n$ dans $\mathbb{R}$ où n équivaut au moins à l'arité de la contrainte associée. Mais il est aussi courant, en conception, de prendre en compte des résultats expérimentaux sous forme d'abaques [6] [2]. L'intégration de ce type de connaissances dans un *CSP* peut se faire à l'aide d'approximation par des fonctions numériques continues et définies par morceaux [25].

CoFiADe permet la définition et le traitement de tables de compatibilité discrètes, continues et mixtes. Il autorise aussi la définition d'expressions mathématiques telles que les fonctions numériques et les contraintes par morceaux.

2.3 Moteur de filtrage

CoFiADe est un outil interactif d'aide à la décision en conception et à la configuration. Contrairement à *Ilog CP*, *Choco* et *ECLiPSe*, Cofiade n'intégre que des méthodes de filtrage : il ne fait pas de résolution de problème. Le moteur de propagation de *CoFiADe*, présenté par l'algorithme 1, permet de répercuter sur l'ensemble des variables du modèle, *via* les contraintes, chaque choix de conception. Chaque choix correspond à une réduction du domaine d'une variable fournie par l'utilisateur. Tout d'abord, grâce aux contraintes d'activation, nous incluons dans le modèle courant les variables, les contraintes et les groupes qui ne l'étaient pas auparavant. La fonction CA retourne l'ensemble des variables nouvellement actives qui sont ajoutées à l'ensemble des variables réduites sur lesquelles le filtrage a lieu.

Alg. 1 MOTEUR(CSP : P)

– $\therefore$ – ***Cet algorithme présente le fonctionnement général du moteur de propagation.***
– $\therefore$ – P $= (\mathbb{V}, \mathbb{D}, \mathbb{C}_C, \mathbb{C}_A)$
– $\therefore$ – Var_{filt} *: ensemble des variables à considérer pour filtrer les contraintes*
$Var_{filt} \leftarrow \emptyset$
– $\therefore$ – Var_{val} *: ensemble des variables actives nouvellement valuées (domaine réduit à un singleton)*
$Var_{val} \leftarrow \emptyset$
Répéter
  – $\therefore$ – *Présentation de l'ensemble des variables actives à valuer à l'utilisateur*
  – $\therefore$ – *Réduction d'une variable v par l'utilisateur D'_v*
  $D_v \leftarrow D_v \cap D'_v$
  Si (D_v est réduit à un singleton) **Alors**
    – $\therefore$ – *v est nouvellement valuée*
    $Var_{val} \leftarrow v$
  Fin Si
  – $\therefore$ – *La variable doit être considérée pour le filtrage*
  $Var_{filt} \leftarrow v$
  Répéter
    – $\therefore$ – *Propagation sur les contraintes d'activation actives pour inclure les nouveaux élèments*
    – $\therefore$ – *Prise en compte dans le problème courant des variables nouvellement activées par ajout de celles-ci à l'ensemble Var_{filt}*
    $Var_{filt} \leftarrow (Var_{filt} \cup \text{CA}(Var_{val}))$
    – $\therefore$ – *Filtrage sur les contraintes de compatibilité actives pour toutes les variables réduites*
    $Var_{val} \leftarrow \text{CC}(Var_{filt})$
  Jusqu'à (variables actives réduites $\neq \emptyset$)
Jusqu'à (il ne reste plus de variables actives à valuer)

Puis, différentes méthodes de filtrage sont appliquées suivant les types de contraintes de compatibilité. *CoFiADe* intègre donc plusieurs méthodes de filtrage de faible degré :

- un algorithme de filtrage par arc-cohérence pour les tables de compatibilité discrètes [20], continues [2] et mixtes [21],
- un algorithme de filtrage par $2B$-cohérence [22], basé sur l'arithmétique des intervalles [23] pour les fonctions numériques et étendu aux multi-intervalles [2],
- un algorithme de filtrage par arbre quaternaire [24] et arbre quaternaire étendu pour les contraintes par morceaux [25].

La fonction CC retourne l'ensemble des variables actives nouvellement réduites.

3 Exemple illustratif

Nous illustrons dans cet exemple tiré de [12] et [1], différentes spécificités de *CoFiADe* : les variables continues avec des domaines de définition multi-intervalles, les tables de compatibilités mixtes et l'activation de variables et contraintes. Puis, nous dérouleront deux scenarii de test illustrant les possibilités de filtrage du moteur de *CoFiADe*. Le modèle présenté ici est disponible à l'url suivante `http://cofiade.enstimac.fr/cgi-bin/cofiade.pl` en choisissant le modèle COFIADE.

3.1 Modèle

Notre exemple illustratif correspond à la configuration et au dimensionnement d'une cuve de mixeur. Il comporte

- quatre variables :
  - une variable discrète *Géométrie_Cuve* caractérisant la géométrie de la cuve du mixeur, toujours présente dans le problème :
    variable symbolic Vessel in {"cylindric", "cubic", "spheric"};
  - trois variables continues :
    - la variable *Volume* caractérisant le volume de la cuve, toujours présente dans le problème et ayant un domaime multi-intervalle :
      variable float Volume in {[0, 27], [28, 150]};
    - la variable *Hauteur* caractérisant la hauteur de la cuve, toujours présente dans le problème :
      variable float Hauteur in {[1,3]};
    - la variable *Rayon*, présente uniquement si la cuve est cylindrique et donc *inactive* au début du problème :
      inactive *variable float Rayon in {[3,4]};*
- une contrainte de compatibilité mixte *ccm* de type table liant la géométrie de la cuve à son volume, toujours présente dans le problème :

```
constraint ccm using (Geometrie_cuve, Volume)
    table {
          {"cylindric", [28, 150]},
          {"spheric", [0.6, 15]},
          {"cubic", [1, 27]},};
```

- trois contraintes de compatibilité continue de type fonction numérique liant le volume, la hauteur et pour l'une d'entre elles, le rayon. Ces contraintes sont inactives au début du problème de configuration et ne sont donc pas filtrées :
  - ***freeze*** *constraint ccc1 using (Hauteur,Volume) condition {Volume = Hauteur * Hauteur * Hauteur * 4 / 3 * 3.14 / 8};*
  - ***freeze*** *constraint ccc2 using (Rayon, Hauteur,Volume) condition {Volume = Rayon * Rayon * 3.14 * Hauteur};*
  - ***freeze*** *constraint ccc3 using (Hauteur,Volume) condition {Volume = Hauteur * Hauteur * Hauteur};*
- trois contraintes d'activation de type table liant la géométrie de la cuve à la fonction de calcul de son volume et pour l'une d'entre elles, au rayon :
  - *constraint cca1 using (Geometrie_cuve) activate (ccc1) table {{"spheric"},};*
  - *constraint cca2 using (Geometrie_cuve) activate (ccc2, Rayon) table {{"cylindric"},};*
  - *constraint cca3 using (Geometrie_cuve) activate (ccc3) table {{"cubic"},};*

3.2 Scenarii de test

L'aide à la décision interactive consiste à valuer ou réduire progressivement le domaine des variables. Lorsqu'une variable v est réduite, la modification de son domaine est propagée aux autres variables via les contraintes. Les valeurs qui ne sont plus compatibles avec le nouveau domaine de v sont alors retirées du domaine des variables. Les contraintes n'étant pas orientées, l'utilisateur peut soit commencer par donner la géométrie de la cuve et en déduire le volume, soit fixer un volume maximal pour en déduire une géométrie de cuve. Nous illustrons ici ces deux modes de fonctionnement.

Déduction du volume Dans le premier scénario, l'utilisateur va tout d'abord fixer la géométrie de sa cuve à {cylindric}. Ce choix a pour conséquence :

- l'activation de la variable *Rayon* qui est ajoutée au problème courant,
- l'activation de la contrainte de calcul du volume *ccc*2 liant le volume, la hauteur et le rayon,
- le filtrage de la contrainte *ccm* par arc-cohérence réduisant le domaine de la variable *Volume* à {[28,

150]},
- le filtrage de la contrainte *ccc*2 par $2B$-cohérence réduisant le domaine de la variable *Volume* à {[28.26, 150]}.

Si l'utilisateur limite la hauteur de cuve à 1.5, *Hauteur* < 1.5, le volume de la cuve est réduit via la contrainte *ccc*2 à $\{[28.26, 75.36]\}$. S'il fixe maintenant le rayon de sa cuve à 3.75, le volume possible est réduit à $\{[44.15, 66.23]\}$. Pour une hauteur de 1.25, le volume final de la cuve est de 55.19.

Déduction de la géomérie Dans le second scénario, l'utilisateur va tout d'abord limiter son volume de cuve à $\{> 30\}$. Ce choix a pour conséquence :

- la valuation de la variable *Géométrie_cuve* à {cylindric} via la contrainte *ccm*,
- l'activation de la variable *Rayon* qui est ajoutée au problème courant,
- l'activation de la contrainte de calcul du volume *ccc*2 liant le volume, la hauteur et le rayon,
- le filtrage de la contrainte *ccc*2 par $2B$-cohérence réduisant le domaine de la variable *Volume* à {]30, 150[}.

Si l'utilisateur limite la hauteur de cuve à 1.25, *Hauteur* < 1.5, le volume de la cuve est reste inchangé et égal à $\{[30, 75.36]\}$. S'il fixe maintenant le rayon de sa cuve à 3.75, le volume possible est réduit à $\{[44.15, 66.23]\}$. Pour une hauteur de 1.25, le volume final de la cuve est de 55.19.

4 Interêts et améliorations prochaines

CoFiADe a servi de support au projet européen *VHT*[4], au projet ANR *ATLAS*[5] et au projet FUI *Helimaintenance*[6] et a donné entière satisfaction en termes :

- de couverture des besoins de modélisation de problèmes de conception (définition de domaines multi-intervalles, de contraintes mixtes et par morceaux, de contraintes d'activation de variables, contraintes et sous-problèmes),
- de facilité de modélisation du problème sous forme de *CSP*,
- de propagation des choix utilisateur (rapidité de traitement, réduction des domaines et de l'espace de solutions).

Grâce à l'interfaçage avec un outil d'optimisation évolutionnaire [26], *CoFiADe* intègre maintenant la possibilité d'optimiser une pré-solution après avoir renseigné un sous-ensemble de variables non négociables et fixé des seuils sur certains critères (tels que le coût ou le délai de fabrication). Cette fonctionnalité récente n'est pas encore accessible en ligne.

À court terme, *CoFiADe* va offrir la possibilité de saisir et mettre en ligne de modèles *CSP*, via une interface Web de saisie de modèles, avec accès limité par mot de passe. *CoFiADe*, manipulant des variables continues, est confronté à des problèmes de stabilité numérique. L'implémentation d'un système de « pas numérique » permettant d'arrêter les itérations lorsque la différence entre les valeurs d'entrée et les valeurs de filtrage sont inférieures à un certain seuil sera bientôt réalisée [27]. Une fonctionnalité de conseil est actuellement en cours de définition et développement dans le cadre des travaux de thèse d'A. Codet de Boisse [28]. Celle-ci permettra de conseiller l'utilisateur sur ses choix à partir d'une base de cas stockant les configurations passées et de lui indiquer les valeurs les plus souvent choisies au vue de la solution courante.

À plus long terme, *CoFiADe* intégrera des contraintes globales telle que *AllDif* [29] [30] et des quantifieurs $\exists$ et $\forall$ afin d'offrir de plus large possibilité de modélisation. Afin d'améliorer le filtrage de *CoFiADe* sur certaines parties spécifiques du problème, son couplage avec des moteurs externes est aussi envisagé. Enfin, afin d'offrir de nouvelles fonctionnalités à l'utilisateur, *CoFiADe* intégrera la notion de préférence [31].

4. Virtual Heat Treatment, projet n° G1RD-CT-2002-00835
5. Aides et assisTances pour la conception, la conduite et leur coupLage par les connAissanceS, projet n° ANR-07-TLOG-002
6. projet pôle de compétitivité Aerospace Valley financé par FUI-DGE

Références

[1] T. van Oudenhove de Saint-Géry. Contribution à l'élaboration d'un formlisame gérant la pertinence pour les problèmes d'aide à la conception à base de contraintes. *Mémoire de doctorat*, Institut National Polytechnique de Toulouse, 2006.

[2] É. Vareilles. Conception et approches par contraintes : contribution à la mise en oeuvre d'un outil d'aide interactif. *Mémoire de doctorat*, Institut National Polytechnique de Toulouse, 2005.

[3] D. Sabin and E.C. Freuder. Configuration as Composite Constraint Satisfaction, *Artificial Intelligence and Manufacturing Research Planning Workshop*, pp. 153-161, 1996.

[4] T. Soininen, T. Tiihonen, T. Männistö and Sulonen R. Towards a General Ontology of Configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 12, n°4, pp. 357-372, 1998.

[5] X. Fischer. Statégie de conduite du calcul pour l'aide à la décision en conception mécanique intégrée : application aux appareils à pression *Thèse*

de doctorat,École Nationale Supérieure d'Arts et Métiers 2000.

[6] T. Mulyanto. Utilisation des techniques de programmation par contraintes pour la conception d'avions *Thèse de doctorat* Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, France, 2002.

[7] K. Hadj-Hamou. Contribution à la conception de produits à forte diversité et de leur chaîne logistique : une approche par contraintes *Thèse de doctorat*, Institut National Polytechnique de Toulouse, 2002.

[8] Y. Vernat. Formalisation et qualification de modèles par contraintes en conception préliminaire *Thèse de doctorat*, École Nationale des Arts et Métier, Bordeaux, 2004.

[9] S. Giaccobi. Méthode de conception de multimatériaux à architecture multicouche : application à la conception d'une canalisation sous-marine *Thèse de doctorat*, Université de Bordeaux, 2009.

[10] M. Djefel. Couplage de la configuration de produit et de projet de réalisation : exploitation des approaches par contraintes et des algorithmes évolutionnaires *Thèse de doctorat*, Institut National Polytechnique de Toulouse, 2011.

[11] J. Abeille. Vers un couplage des processus de conception de systèmes et de planification de projets : formalisation de connaissances méthodologiques et de connaissances métier *Thèse de doctorat*, Institut National Polytechnique de Toulouse, 2011.

[12] E. Gelle. On the generation of locally consistent solution spaces in mixed dynamic constraint problems *Thèse de doctorat* École Polytechnique Fédérale de Lausanne, Suisse, 1998.

[13] J-F. Puget. A C++ Implementation of CLP, *Proceedings of SPICIS 94*, November 1994, Singapore.

[14] CHOCO Team. Choco : an Open Source Java Constraint Programming Library, *Research report*, Ecole des Mines de Nantes, `http://www.emn.fr/z-info/choco-solver/pdf/choco-presentation.pdf`, 2010.

[15] M. Meier and J. Schimpf, *An Architecture for Prolog Extensions*, Proceedings of the 3rd International Workshop on Extensions of Logic Programming, Bologna, 1992.

[16] U. Montanari. Networks of constraints : fundamental properties and application to picture processing *Information sciences*, Vol(7), pp. 95-132, 1974.

[17] J.C. Régin. Amélioration de l'expressivité des contraintes de table. *Journées francophones de programmation par contraintes* JFPC 2011, pp 281-287, 2011.

[18] E. Tsang. Foundations of Constraints Satisfaction. *Academic Press*, London, 1993.

[19] S. Mittal and B. Falkenhainer. Dynamic Constraint Satisfaction Problems *AAAI* Boston, US, pp. 25-32, 1990.

[20] A.K. Mackworth, *Consistency in Networks of Relations*, Artificial Intelligence, pp. 99-118, 1977.

[21] B. Faltings. Arc Consistency for Continuous Variables, *Artificial Intelligence*, vol(65), pp. 363-376, 1994.

[22] O. Lhomme. Consistency techniques for numeric *CSP*, *International Joint Conference on Artificial Intelligence*, pp. 232-238, 1993.

[23] R.E. Moore. Interval Analysis, *Prentice-Hall*, 1966.

[24] D. Sam-Haroud and B. Faltings. Consistency Techniques for Continuous Constraints, *Constraints Journal*, pp. 85-118, 1996.

[25] É. Vareilles, M. Aldanondo and P. Gaborit. How to take into account piecewise constraints in constraint satisfaction problem, *Engineering Applications of Artificial Intelligence*, Elsevier, 2009.

[26] P. Pitiot, M. Aldanondo, É. Vareilles, P. Gaborit, M. Djefel and S. Carbonnel. Concurrent product configuration and process planning, towards an approach combining interactivity and optimality, *International Journal of Production Research*, Accepted, 2011.

[27] O. Lhomme and M. Rueher. Application des techniques CSP au raisonnement sur les intervalles, *revue d'Intelligence Artificielle*, 11(3) :283-311, 1997.

[28] É. Vareilles, M. Aldanondo, A. Codet de Boisse, T. Coudert, P. Gaborit and L. Geneste. How to take into account general and contextual knowledge for interactive aiding design : towards the coupling of CSP and CBR approaches. *Engineering Applications of Artificial Intelligence, EAAI*, DOI : 10.1016/j.engappai.2011.09.002, 2011.

[29] J.C. Régin. A Filtering algorithm for constraints of difference in csps. *AAAI*, pages 362 :367, 1994.

[30] N. Beldiceanu and S. Demassey. Global constraint catalog, `http://www.emn.fr/x-info/sdemasse/gccat/index.html`, 2009.

[31] H. Fargier, J. Lang and T. Schiex. Selecting preferred solutions in Fuzzi Contraint Satisfaction Problems. *Proc. of the* 1^{st} *European Congress on Fuzzy and Intelligent Technologies*, 1993.

Sélection adaptative d'opérateurs pour la recherche locale basée sur un compromis exploration-exploitation

Nadarajen Veerapen[1] **Jorge Maturana**[2] **Frédéric Saubion**[1]

[1] LUNAM Université, Université d'Angers, LERIA, Angers, France
[2] Instituto de Informática, Universidad Austral de Chile, Valdivia, Chili
prenom.nom@univ-angers.fr prenom.nom@inf.uach.cl

Résumé

Cet article traite de la sélection adaptative d'opérateurs dans le contexte de la recherche locale (LS). La similarité entre la solution candidate et les solutions de la trajectoire de recherche est considérée. De paire avec la qualité de la solution, ces deux mesures nous permettent d'évaluer la performance de chaque opérateur. Une nouvelle mesure d'utilité pour opérateurs de LS, basée sur les distances relatives entre les opérateurs, est proposée. En utilisant des méthodes basiques de sélection, nous la comparons à une mesure proposée récemment basée sur le concept de dominance Pareto. Une version adaptative de l'algorithme est également examinée (variation du biais exploration-exploitation). Les méthodes proposées sont testées sur le problème d'affectation quadratique et le problème du voyageur de commerce asymétrique.

1 Introduction

Dans cet article [1] nous nous intéressons à la recherche locale (LS) pour l'optimisation combinatoire. D'une façon générale, un algorithme de recherche locale se déplace d'une configuration à une autre, en construisant un chemin de recherche. Ces mouvements peuvent être exécutés en utilisant des opérateurs qui sélectionnent la prochaine configuration à atteindre dans le voisinage de la configuration courante. Une politique de recherche efficace requiert un compromis entre deux objectifs généralement divergents : l'exploitation et l'exploration. L'exploitation implique la convergence vers un optimum local tandis que l'exploration implique un échantillonnage adéquat de l'espace de recherche. Ce compromis peut être atteint en contrôlant les opérations de bases (mouvements ou opérateurs) qui dirigent le chemin de recherche dans l'espace de recherche.

La conception d'algorithmes de recherche autonomes est devenue un domaine important de la résolution de problèmes [11]. Au cours des dernières décennies, un nombre croissant de techniques de résolution a été proposé dans l'optique de résoudre des problèmes de plus en plus complexes. Toutefois elles sont souvent difficiles à adapter et à paramétrer pour un problème donné. En fait, les outils efficaces de résolution ne sont plus à la portée des utilisateurs. La conception de solveurs toujours plus efficaces produit fréquemment des systèmes fortement complexes, qui requièrent une somme non-négligeable de connaissance d'expert, par exemple pour en sélectionner intelligemment les paramètres. Cette tendance mène donc vers un besoin croissant pour des algorithmes de configuration automatique de paramètres tels que [12, 7, 13, 3]. La configuration automatique est exécutée hors-ligne en faisant tourner l'algorithme sur des instances de test. Il est également possible de contrôler les paramètres d'un algorithme en ligne, par exemple dans le contexte de la recherche réactive [1], en intégrant des techniques d'apprentissage aux heuristiques de recherche locale. La conception de stratégies de contrôle de haut niveau est une tendance relativement récente ayant pour but de rendre les techniques d'optimisation plus abordables [5]. Une classification de ces approches est présentée dans [10].

Nous nous concentrons ici sur la sélection en ligne du prochain mouvement à appliquer dans un processus de recherche locale afin de s'adapter à la réalité du

1. Cet article est une traduction de l'article *An Exploration-Exploitation Compromise-Based Adaptive Operator Selection for Local Search*, GECCO'12 [26].

sous-ensemble de l'espace de recherche en cours d'exploration. Adaptive Pursuit [23] est un exemple d'une telle stratégie affectant une probabilité de sélection à chaque opérateurs pour les algorithmes génétiques. Sa caractéristique principale est l'utilisation du concept du gagnant remporte tout (winner-takes-all) qui se traduit par la forte augmentation de la probabilité de sélection du meilleur opérateur tout en diminuant la probabilité de sélection des autres opérateurs. De nouvelles techniques de sélection adaptative d'opérateurs ont été proposées pour les algorithmes évolutionnaires, par exemple dans [8].

Dans cet article nous présentons un mécanisme de sélection d'opérateur pour la recherche locale. Un opérateur est une combinaison d'un voisinage et d'une fonction de sélection. À chaque itération, un opérateur est sélectionné à partir d'un ensemble d'opérateurs. Les critères de sélection sont basés sur la performance antérieure des opérateurs ainsi que sur un compromis entre exploration et exploitation que le processus entend encourager. Nous présentons une méthode, brièvement étudiée dans [25], permettant d'évaluer la performance d'un opérateur basée sur sa performance relative. Le compromis souhaité est tout d'abord fixé. Nous examinons ensuite une stratégie adaptative basée sur la fréquence d'apparition des valeurs des solutions récemment obtenues.

Nos fonctions de contrôle s'adressent à des problèmes de permutation. Ceci nous permet d'utiliser des combinaisons basiques de voisinages et de fonctions de sélections simples tout en obtenant un ensemble d'opérateurs suffisamment varié.

Dans [27], les opérateurs sont sélectionnés sur le principe de la Pareto dominance. La probabilité de sélection d'un opérateur est déterminée en fonction du nombre d'opérateurs que ce même opérateur domine. Une limite de cette méthode est qu'il n'y a pas de compromis explicite entre exploration et exploitation, ce qui favorise les opérateurs au milieu du front Pareto. Cet article a pour but d'étudier une stratégie alternative basée sur un compromis explicite et se base sur les résultats initiaux présentés dans [25].

Cet article est composé de sept sections. Après cette introduction, la section 2 présente les définitions et les concepts importants : voisinage, fonction de sélection et opérateur. La section 3 décrit l'intégration de la sélection d'opérateurs dans un algorithme LS ainsi que les métriques utilisées. La section 4 explique la sélection d'opérateurs. Elle comprend également un survol des travaux connexes. Les expériences pour un compromis statique sont présentées en section 5. La section 6 quant à elle est dédiée au expériences avec un compromis adaptatif. Enfin la dernière section présente nos conclusions.

2 Concepts généraux

Étant donnés un espace de recherche $\mathcal{S}$ et une fonction objectif *eval* sur $\mathcal{S}$, une recherche locale applique des opérateurs afin d'atteindre une solution optimale. Un opérateur, étant donné une configuration de $\mathcal{S}$, produit une nouvelle configuration. Dans sa forme la plus simple, il est composé d'un voisinage et d'une fonction de sélection. Ce voisinage est l'ensemble des configurations atteignables à partir d'une configuration donnée. La fonction de sélection choisit une configuration à partir de cet ensemble en fonction de certains critères. Le squelette principal d'une métaheuristique de recherche locale est l'application d'un opérateur au sein d'une simple boucle. L'opérateur explore un voisinage et renvoie une solution. Selon le type de métaheuristique utilisée, une étape supplémentaire telle qu'une perturbation est introduite dans la boucle.

Les concepts de voisinage, de fonction de sélection et d'opérateur sont formalisés ci-après.

Définition. Soit $\mathcal{S}$ l'espace de recherche. Une *relation de voisinage* ou *voisinage* est une relation binaire irréflexive $\mathcal{N} \subseteq \mathcal{S}^2$ sur l'espace de recherche. Dans la majorité des cas, cette relation est symétrique.

La composition et l'union de voisinages sont notées $\mathcal{N} \circ \mathcal{N}'$ et $\mathcal{N} \cup \mathcal{N}'$ respectivement. Un voisinage composé avec lui-même est noté $\mathcal{N}^2$ et $\mathcal{N}^{n+1} = \mathcal{N} \circ \mathcal{N}^n$.

Considérons le paysage de recherche. La relation d'ordre $<$ sur $\mathcal{S}$ correspond à l'ordre induit par la fonction objectif du problem. Seuls les problèmes de minimisation sont considérés sans perte de généralité.

Définition. Un *sélecteur* est une fonction qui effectue une sélection sur le voisinage, éventuellement guidé par l'ordre $<$ et défini par $\sigma : \mathcal{S} \times 2^{\mathcal{S}^2} \mapsto \mathcal{S}$ (ici la sélection ne retourne qu'un voisin), tel que $(s, \sigma(s, \mathcal{N})) \in \mathcal{N}^=$ (la clôture réflexive de $\mathcal{S}$ afin d'inclure l'identité).

Définition. Un *opérateur* est alors défini par une paire $(\mathcal{N}, \sigma)$.

3 Contrôle d'opérateurs pour la LS

Le contrôle d'opérateurs se réfère à la sélection d'un opérateur approprié à partir d'un ensemble d'opérateurs disponibles à chaque itération d'un algorithme de LS. Le comportement antérieur de chaque opérateur est enregistré et analysé afin de déterminer son utilité par rapport à la gestion du compromis entre exploitation et exploration. Le schéma 1 dépeint l'interaction entre le contrôleur et la LS. Après chaque application d'un opérateur, le contrôleur reçoit la qualité de la configuration courante ainsi que sa distance

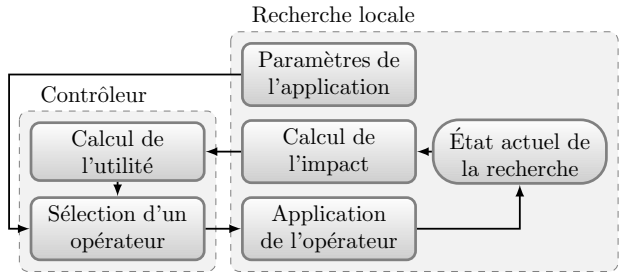


FIGURE 1 – Contrôleur et algorithme LS

par rapport au chemin de recherche. Ces métriques représentes l'impact de l'opérateur et seront décrite plus loin dans cette section. L'impact est utilisé pour calculer et mettre à jour l'utilité de chaque opérateur. Le contrôleur sélectionne ensuite le prochain opérateur à partir de ce critère. L'utilité et le processus de sélection sont décrits dans la section 4.

Quand un utilisateur souhaite résoudre un nouveau problème, il n'a qu'à définir une procédure pour lire les données de l'instance et spécifier la fonction objectif ainsi que les contraintes. L'utilisateur peut également ajouter de nouveaux opérateurs à l'ensemble d'opérateurs prédéfinis. Il est donc primordial que le contrôle d'opérateur soit suffisamment robuste pour gérer des opérateurs peu performants.

3.1 Calcul de l'impact

Comme précédemment mentionné, un aspect fondamental de la LS est le compromis entre exploitation et exploration. Des métriques adéquates sont donc nécessaires afin d'évaluer ces deux notions liées à la qualité et à la diversité respectivement. Nous utilisons ici des métriques de [27].

3.1.1 Qualité

La qualité est directement mesurée à partir de la fonction objectif. Le changement relatif de la qualité lorsqu'un opérateur *op* est appliqué à une solution s est donné par

$$q = \frac{eval(s) - eval(op(s))}{eval(s) + 1} \quad (1)$$

3.1.2 Distance

Mesurer la diversité est relativement simple pour les algorithmes à populations mais moins clair pour les algorithmes qui ne sont basés que sur une solution. Ici nous considérons une fenêtre glissante contenant les solutions antérieures du chemin de recherche et nous mesurons la différence entre ces solutions et la solution candidates $c = op(s)$. Cette différence est calculée au niveau du couple variable-valeur : moins fréquentes sont les occurrences des couples variable-valeur de la solution candidate dans le chemin, plus grande est la distance est entre eux. L'équation suivante formalise cette notion. Soit $P_{i,j}$ le chemin de l'itération i jusqu'à j, $i \leq j$. Alors

$$d^P(c, P_{i,j}) = \frac{1}{n} \times \sum_{k_c=1}^{n} \left(1 - \frac{occ(P_{i,j}, (k_c, \pi_{k_c}))}{|P_{i,j}|}\right) \quad (2)$$

où $occ(P_{i,j}, (a, b))$ correspond au nombre de fois où le couple variable-valeur (a, b) apparaît dans $P_{i,j}$. Par exemple dans $(1, 3, 2)$, les couples sont $(1, 1)$, $(2, 3)$ et $(3, 2)$. Pour des problèmes qui modélisent des cycles, (a, b) représente deux valeurs consécutives. Par exemple dans $(1, 3, 2)$ les couples sont $(1, 3)$, $(3, 2)$ et $(2, 1)$.

Maintenant que nous avons défini comment mesurer la qualité et la distance par rapport aux solutions, la section suivante traite du calcul de l'utilité de chaque opérateur en fonction de ces mesures.

4 Sélection d'opérateurs

Dans cet article, un opérateur est sélectionné proportionnellement à valeur d'utilité représentant sa performance antérieure.

Une fenêtre glissante de taille fixe, enregistrant les valeurs de qualité et de distance sur les m dernières applications, est associée à chaque opérateur. Ces fenêtres sont initialisées en appliquant une fois chaque opérateur au début de la recherche. L'utilité d'un opérateur est calculée en se basant sur la moyenne des valeurs de qualité et de distance. Dans la suite, lorsque la qualité ou la distance d'un opérateur sera mentionnée, nous nous référerons implicitement à la valeur moyenne sur la fenêtre glissante. Nous définissons les différentes mesures d'utilité dans la sous-section suivante.

4.1 Utilité d'un opérateur

Dans [27] l'utilité de chaque opérateur est calculée en fonction du nombre d'opérateurs qu'il domine.

Définition. Soient deux vecteurs u et v de même cardinalité p et considérant un problème de maximisation, u *domine* u si $u_k \geq v_k, \forall k \in \{1, \ldots, p\}$ avec au moins une inégalité stricte. Ceci est souvent appelé la Pareto dominance et notée $u \succ v$.

L'utilité basée sur la Pareto dominance U_P de l'opérateur o dans l'ensemble O de n opérateurs est alors

définie par

$$U_P(o) = |\{o' | o' \in O, o \succ o'\}| + \epsilon \quad (3)$$

où ϵ est une valeur positive afin d'obtenir une utilité non nulle.

Cette utilité ne permet pas de spécifier un compromis entre exploitation et exploration explicitement. Nous introduisons une pondération, α, afin d'influencer ce compromis.

Soient deux opérateurs o_1 and o_2 définis par les couples qualité-distance (q_1, d_1) et (q_2, d_2), nous définissons le déplacement rectilinéaire pondéré de o_1 vers o_2 tel que

$$d_\alpha(o_1, o_2) = \alpha(d_1 - d_2) + (1 - \alpha)(q_1 - q_2) \quad (4)$$

À partir de cette métrique, nous définissons la nouvelle utilité : somme des déplacements positifs vers les autres opérateurs.

$$U_\alpha^{\Sigma+}(o) = \sum_{i=1}^{n} max(0, d_\alpha(o, o_i)) \quad (5)$$

Comme point de comparaison, nous définissons également la simple somme pondérée de la qualité et de la distance.

$$U_\alpha(o) = \alpha d + (1 - \alpha)q \quad (6)$$

La figure 2 tente d'illustrer l'intérêt d'utiliser la somme des déplacements comme valeur d'utilité d'un opérateur. Primo, les déplacements, et leur somme, sont utiles puisqu'ils décrivent quantitativement (la magnitude) et, dans une moindre mesure, qualitativement (le signe ou la direction) la relation entre opérateurs. Secundo, la pondération introduit naturellement un biais quantifiable vers l'exploration ou l'exploitation.

4.2 Travaux connexes

Parmi un grand nombre de travaux [11], la conception d'algorithmes de recherche locale autonome a été traitée dans le contexte de la recherche réactive [1] dans des travaux fondateurs tels que le tabou réactif [2], le recuit simulé adaptatif [14] ou la recherche locale adaptative [6]. En général seul un critère, la qualité de la solution, est considéré. Ces travaux se concentrent sur des paramètres correspondants aux heuristiques spécifiques de ces méthodes, par exemple la taille de la liste tabou.

Dans cet article, nous souhaitons manipuler simultanément le voisinage et la fonction de sélection en tant qu'entité unique : l'opérateur. De plus, nous souhaitons également utiliser de multiples critères génériques (la qualité et la diversité) afin d'orienter la politique de recherche.

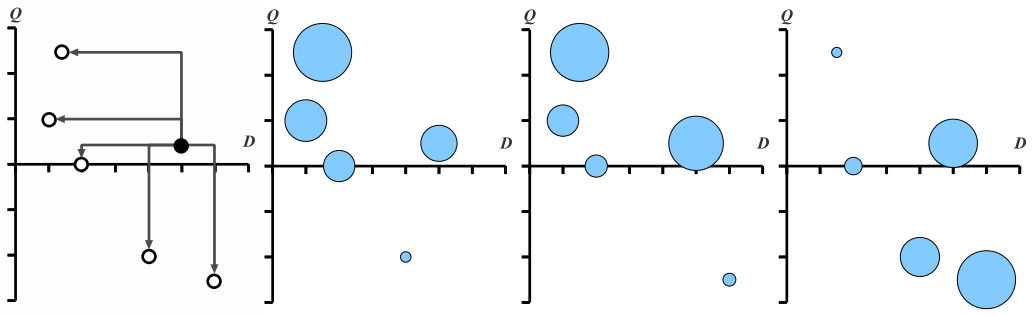
En ce qui concerne le contrôle de paramètres. les techniques les plus avancées ont été initialement développées dans le contexte de l'optimisation évolutionnaire [15]. Comme mentionné précédemment, Adaptive Pursuit [23] affecte une probabilité à chaque opérateur pour les algorithmes génétiques en utilisant une stratégie du gagnant remporte tout (winner-takes-all).

Dans sa thèse de doctorat, Fialho [8] décrit un nombre de stratégies de sélection (sélection adaptative d'opérateurs) plus récentes pour les algorithmes génétiques, dont le bandit manchot multibras dynamique qui sélectionne un opérateur qui maximise la somme de deux valeurs. La première représente la performance de l'opérateur et la seconde assure que l'opérateur sera sélectionné un nombre infini de fois. Il présente également plusieurs mesures pour déterminer l'utilité des opérateurs tel que l'AUC (l'aire sous la courbe) pour un critère (la qualité de la population). Dans [19, 18], des techniques de sélection d'opérateurs sont proposées afin de traiter deux critères simultanément dans l'évaluation des opérateurs : la qualité et la diversité de la population. Dans le contexte de la recherche locale, ceci a également été tenté dans [27], offrant des résultats préliminaires.

Plus récemment, un solveur fortement paramétrable pour SAT a été présenté dans [24] où la probabilité d'utiliser un voisinage ou une fonction de sélection est déterminée par un réglage hors-ligne intensif. Si nous considérons la LS associée au paradigme de programmation par contrainte, [17] présente un concept apparenté à la sélection adaptative d'opérateurs pour la recherche à grand voisinage pour le problem d'ordonnancement de véhicules. Des résultats initiaux pour une version améliorée basée sur l'apprentissage par renforcement sont présentés dans [16]. Par rapport à ces précédentes approches en LS, nous avons comme objectif de proposer des méthodes plus sophistiquées pour l'évaluation de l'utilité des opérateurs ainsi que des techniques de sélection plus adaptatives. Toutefois, vue la spécificité à un problème donné des méthodes précédemment citées, la comparaison avec notre méthode est problématique. Nous entendons traiter cet aspect dans un prochain travail, en élargissant la gamme de problèmes traités.

5 Expériences pour l'utilité pondérée

Le codage des configurations correspond à un problème de permutation. Si N est la taille du problème (le nombre de variables), une solution est représentée par une permutation de N entiers $\{1, 2, \ldots, N\}$, c.-à-d. que chaque variable a le même domaine et deux va-



(a) Déplacement (b) $\alpha = 0.0$ (c) $\alpha = 0.5$ (d) $\alpha = 1.0$

FIGURE 2 – (a) Une population hypothétique de six opérateurs montrant les déplacements non pondérés relativement à un opérateur ; (b)–(d) α détermine l'importance (taille de la bulle) de chaque opérateur pour $U_{\alpha}^{\Sigma+}$.

riables ne peuvent avoir la même valeur.

Soit Π l'espace de recherche, c.-à-d. l'ensemble de toutes les permutations de l'ensemble $\{1, 2, \ldots, N\}$. Si $\pi \in \Pi_N$ et $1 \leq i \leq N$, alors π_i correspond à l'élément i dans π.

Les différentes mesures d'utilité décrites dans la section précédente sont tout d'abord testées sur un problème classique de permutation : le problème d'affectation quadratique (QAP) dans lequel il faut trouver le coût minimum lorsque que N unités de productions sont placés dans N lieux, le coût étant la somme de tous les produits distance-flot possibles. La fonction objectif est donnée par

$$min_{\pi \in \Pi_N} \sum_{i=1}^{N} \sum_{j=1}^{N} a_{\pi_i, \pi_j} b_{i,j} \tag{7}$$

Afin d'évaluer la robustesse des méthodes proposées, des tests sont ensuite basés sur le problème du voyageur de commerce asymétrique (ATSP) qui revient à trouver le plus court circuit entre N villes (le plus petit cycle hamiltonien) étant donné la distance $d(c_i, c_j)$ pour chaque paire de villes (c_i, c_j). La fonction objectif est donnée par

$$min_{\pi \in \Pi_N} \sum_{i=1}^{N-1} d(c_{\pi_i}, c_{\pi_{i+1}}) + d(c_{\pi_N}, c_{\pi_1}) \tag{8}$$

Les expériences sur l'ATSP sont décrites à la section 6.2.

5.1 Cadre expérimental

Chaque opérateur est une combinaison d'un voisinage et d'une fonction de sélection. Nous utilisons un voisinage de base, $\mathcal{N}_E$ (échange). Il est défini tel que $(s, s') \in \mathcal{N}_E$ si et seulement si $s = (\pi_1, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j, \pi_{j+1}, \ldots, \pi_n)$ et $s' = (\pi_1, \ldots, \pi_{i-1}, \pi_j, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_i, \pi_{j+1}, \ldots, \pi_n)$ pour i et j quelconques. De plus grands voisinages sont créés par composition de ce voisinage avec lui-même.

Le tableau 1 présente les différents sélecteurs et opérateurs utilisés. Parmi les 12 opérateurs, 10 d'entre eux (O1–O10) sont des opérateurs modérés, c.-à-d. qu'ils ne modifient qu'un petit nombre de variables. Les deux derniers (O11 et O12) sont des opérateurs perturbateurs modifiant 25 % et 50 % des variables respectivement. Les opérateurs O1–O5 sont orientés exploitation tandis que les O6–O12 sont orientés exploration. Le mécanisme de sélection d'opérateurs n'a pas connaissance de ces informations.

Les opérateurs perturbateurs sont inclus afin de pénaliser la sélection uniforme et d'augmenter la difficulté de la tâche des mécanismes de sélection. En effet, des travaux précédents [27] ont montrés que si la population d'opérateurs est raisonnablement bien sélectionnée, la sélection uniforme parvient à produire des résultats de qualité raisonnable. De plus, ces opérateurs perturbateurs peuvent être considérés comme des mécanismes de redémarrage dynamiques puisque les opérateurs ne sont pas appliqués dans un ordre prédéterminé.

Tel que mentionné précédemment, les opérateurs sont simplement sélectionnés proportionnellement à leur utilité (noté PR plus loin). Nous comparons cette méthode de sélection à une autre méthode légèrement plus complexe : Adaptive Pursuit [23].

Adaptive Pursuit (AP) emploie une stratégie de gagnant remporte tout. Une probabilité minimum p_{min} est utilisée ainsi qu'un paramètre α' qui contrôle l'importance des valeurs précédemment obtenues. Plus α' est grand, plus la nouvelle valeur a de l'importance. Le

TABLE 1 – Définitions des sélecteurs et des opérateurs

Sélecteur	Nom	Définition
σ_R	Aléatoire	$\sigma_R(s,\mathcal{N})$ est n'importe quel s' tel que $(s,s') \in \mathcal{N}$
σ_I	Amélioration	$\sigma_I(s,\mathcal{N})$ est n'importe quel s' tel que $(s,s') \in \mathcal{N}$ and $s' < s$
σ_{BI}	Meilleure amélioration	$\sigma_{BI}(s,\mathcal{N})$ est un élément minimal s' selon l'ordre $<$, tel que $(s,s') \in \mathcal{N}$
σ_{BIk}	Meilleure amélioration k	$\sigma_{BIk}(s,\mathcal{N})$ est un élément uniformément sélectionné $s' \in K$, K étant l'ensemble des k-meilleurs éléments selon l'ordre $<$, tel que $(s,s') \in \mathcal{N}$
σ_{Tk}	Tournois k	$\sigma_{Tk}(s,\mathcal{N})$ est un élément s' tel que K est le sous-ensemble de k éléments en relation avec s dans $\mathcal{N}$ et s' est le meilleur de ces k éléments
Opérateur	**Nom**	**Définition**
$(\sigma_I, \mathcal{N}_E)$	O1	première amélioration pour l'échange entre deux variables.
$(\sigma_{BI}, \mathcal{N}_E)$	O2	meilleure amélioration pour l'échange entre deux variables.
$(\sigma_{BI5}, \mathcal{N}_E)$	O3	choix aléatoire parmi les cinq meilleurs échanges entre deux variables.
$(\sigma_{BI}, \mathcal{N}_E)^2$	O4	deux meilleurs échanges consécutifs ; les variables échangées à l'étape une sont interdites à l'étape deux.
$(\sigma_{BI}, \mathcal{N}_E)^3$	O5	trois meilleurs échanges consécutifs ; les variables échangées aux étapes précédentes sont interdites à l'étape suivante.
$(\sigma_{T3}, \mathcal{N}_E^2)$	O6	meilleur échange entre trois variables choisies aléatoirement.
$(\sigma_{T4}, \mathcal{N}_E^3)$	O7	meilleur échange entre quatre variables choisies aléatoirement.
$(\sigma_{T5}, \mathcal{N}_E^4)$	O8	meilleur échange entre cinq variables choisies aléatoirement.
$(\sigma_{T6}, \mathcal{N}_E^5)$	O9	meilleur échange entre six variables choisies aléatoirement.
$(\sigma_R, \mathcal{N}_E)^3$	O10	trois échanges consécutifs entre deux variables.
$(\sigma_R, \mathcal{N}_E)^{\lfloor n/8 \rfloor}$	O11	$\lfloor n/8 \rfloor$ échanges aléatoires consécutifs entre deux variables.
$(\sigma_R, \mathcal{N}_E)^{\lfloor n/4 \rfloor}$	O12	$\lfloor n/4 \rfloor$ échanges aléatoires consécutifs entre deux variables.

paramètre β défini le niveau de « gloutonnerie ». Parmi k opérateurs la probabilité de sélectionner l'opérateur i à l'itération t est donnée par

$$p_{i,t} = \begin{cases} p_{i,t-1} + \beta(p_{max} - p_{i,t-1}) & \text{if } i = i^*_{t-1} \\ p_{i,t-1} + \beta(p_{min} - p_{i,t-1}) & \text{otherwise} \end{cases} \quad (9)$$

où

$$i^*_t = \text{argmax}_{i=1\ldots k}\{Q_{i,t}\} \text{ et } p_{max} = 1 - (k-1)p_{min}$$

et

$$Q_{i,t} = (1-\alpha')Q_{i,t-1} + \alpha' U_{i,t-1}$$

L'utilité basée sur la Pareto dominance est utilisée pour AP avec $\epsilon = 0$ puisque p_{min} joue une rôle équivalent.

Le paramètre pour U_P est $\epsilon = 0.1$ (les valeurs 0.01, 0.05, 0.2 et 1 ont également été testées mais 0.1 a obtenue les meilleurs résultats sur une majorité d'instances). Pour $U_\alpha^{\Sigma+}$ et U_α, le paramètre α a été testé avec les valeurs 0.2, 0.5 et 0.8.

La méthode de réglage F-Race [3] a été utilisée pour déterminer la valeur des paramètres de AP ($\alpha' = 0.9$, $\beta = 0.1$, $p_{min} = 0.01$). Les combinaisons de paramètres testées ont été générées à partir de $\{0.1, 0.3, 0.6, 0.9\}$ pour α' et β et $\{0, 0.001, 0.01, 0.1, 0.1\}$ pour p_{min}.

Pour chaque instance, chaque algorithme est lancé 30 fois et un maximum de 40 000 itérations lui sont accordées par exécution. Pour le QAP, les mêmes 30 solutions aléatoires sont utilisées au début de chaque algorithme. Pour l'ATSP la solution initiale est calculée de façon gloutonne. La longueur de toutes les fenêtres glissantes est fixée à 100 (pour le chemin et les mesures de qualité et de distance pour chaque opérateur).

5.2 Analyse des résultats

Le tableau 2 présente les résultats concernant toutes les expériences sur des instances de QAP issues de QAPLIB [4] dont parle cet article. Les valeurs pour chaque algorithme représentent le pourcentage moyen de différence par rapport aux meilleures valeurs connues. Dans cette section nous nous intéressons à toutes les colonnes sauf l'avant-dernière. Les résultats pour U_α (disponibles dans [25]) ne sont pas présentés dans le tableau puisqu'ils n'offrent pas de meilleurs résultats que U_P pour les valeurs α testées. La dernière colonne donne, à titre de comparaison, les résultats obtenus pour le tabou robuste (RTS) [22], un algorithme LS dédié au QAP. RTS est un algorithme relative-

ment ancien mais a été choisi car il demeure à ce jour l'algorithme ayant trouvé le plus grand nombre de meilleures solutions connues pour les instances de QAPLIB n'ayant pas été résolues de façon optimale. Les chiffres en gras indiquent les meilleures valeurs et en italique les résultats à 0.05 % de la meilleure valeur (les résultats de RTS ne sont pas considérés puisqu'ils sont toujours meilleurs ou équivalents).

Considérons tout d'abord la qualité générale moyenne des solutions obtenues avec toutes les méthodes. Toutes, même la sélection uniforme, sont généralement à un ou deux pour cents de l'optimale ou de la meilleure valeur connue. Ceci démontre que le nombre alloué d'itérations n'est pas un facteur limitant.

Le tableau 2 nous permet d'observer que U_P (un paramètre) avec une sélection dont la probabilité est directement calculée à partir de l'utilité (PR dans le tableau) se comporte très bien par rapport à AP qui a trois paramètres. Pour AP, le réglage des paramètres ne lui permet que de se comporter à un niveau équivalent ou pire que PR. Le concept du gagnant remporte tout, utile pour les algorithmes évolutionnaires, ne semble pas apporter de bénéfice dans les conditions de test utilisées ici. Au lieu de donner l'avantage au meilleur opérateur, il semble qu'il soit plus approprié d'avoir une probabilité plus élevée de choisir parmi quelques opérateurs qui se comportent raisonnablement bien.

Si nous comparons U_P (PR) et $U_\alpha^{\Sigma+}$, il en ressort que la seconde est généralement équivalente à la première pour un petit α et vraiment meilleur sur les instances taixxa. À partir de cette observation, et parce que différentes classes d'instances semblent nécessiter différents α pour obtenir les meilleurs résultats, une méthode pour faire varier α et améliorer les résultats est décrite dans la section suivante.

U_α est la seule utilité à ne pas être calculé à partir d'une relation entre opérateurs. Ainsi que mentionné précédemment, ses résultats sont pire que U_P. Il semble donc que l'utilisation d'une simple somme pondérée est trop simpliste et que considérer la relation entre opérateurs peut être utile lorsqu'elle est utilisée de façon appropriée.

Nous avons inclus RTS comme point de comparaison afin d'illustrer les possibilités d'amélioration pour les méthodes génériques. Néanmoins, nous ne nous attendions pas à dépasser un algorithme dédié basé sur des prohibitions en utilisant une méthode générique utilisant des opérateurs génériques pour des problèmes de permutation, dont certains étant très perturbateurs.

La section suivante se penche sur une méthode pour faire varier α au fil de la recherche afin d'obtenir de meilleurs résultats.

FIGURE 3 – Les modules Stratégie, Contrôleur et LS

6 Valeurs adaptatives de paramètres pour le contrôle d'opérateurs

Dans la section précédente nous avons utilisé un compromis statique. Nous examinons maintenant comment ce compromis peut s'adapter en ligne à l'état du processus de recherche.

6.1 Stratégie de réglage des paramètres

Afin de faire varier α, nous introduisons le module Stratégie qui se fixe au contrôleur comme illustré sur le schéma 3. Il communique avec ce dernier et son rôle est de faire varier les paramètres en direct.

Dans cet article nous considérons une stratégie de diversité « correcte »(CD), un concept développé dans [20] pour les algorithmes évolutionnaires. La stratégie CD utilise la qualité des solutions de la population comme moyen d'évaluation de la diversité de la population. Si le nombre de solutions ayant la même qualité est au-dessus d'un certain seuil T_{max} alors on suppose que la population est trop homogène et la diversité commandée est incrémentée d'un pas s_{inc}. De façon symétrique, si le nombre de solutions ayant la même qualité est au-dessous d'un autre seuil T_{min} la diversité commandée est décrémentée d'un pas s_{dec}, la supposition étant que la population est incapable de rester proche d'optima locaux de bonne qualité, ce qui démontre une faible exploitation. Dans notre framework, CD utilise les solutions présentes dans la fenêtre glissante, la même fenêtre qui est utilisée pour le calcul de la distance.

6.2 Cadre expérimental

Nous avons utilisé F-Race afin d'effectuer le réglage des paramètres. Les valeurs testées proviennent des domaines suivants : $T_{max} = \{0.3, 0.4, 0.5, 0.6\}$, $T_{min} = \{0.05, 0.1, 0.15, 0.20, 0.25\}$, and $\{0.0001, 0.001, 0.01, 0.1\}$ for s_{inc} et s_{dec}. Ceci donne 320 combinaisons. Le gagnant est ($T_{max} = 0.3$, $T_{min} = 0.25$, $s_{inc} = 0.0001$, $s_{dec} = 0.1$).

La combinaison gagnante étant à une extrémité de chaque domaine, on peut être amené à penser qu'une meilleure combinaison peut être obtenue en élargissant les domaines. Nous avons donc lancé un deuxième réglage avec de nouveaux domaines et obtenu un nouveau vainqueur ($T_{max} = 0.35$, $T_{min} = 0.15$, $s_{inc} =$

0.0001, $s_{dec} = 0.01$). Ce dernier est relativement différent du précédent mais n'a pas bénéficié des nouvelles valeurs aux extrémités de chaque domaine.

Lorsque les deux gagnants sont testés côte à côte, leurs distributions de résultats sont statistiquement équivalentes. Ceci nous amène à penser que la stratégie n'est pas hypersensible à sa configuration de paramètres et est donc relativement robuste.

En plus du QAP, l'ATSP est également utilisé dans cette section (instances provenant de TSPLIB [21]). Le scénario de test pour l'ATSP réutilise les paramètres trouvés pour le QAP (test de robustesse). De nouvelles combinaisons de paramètres sont ensuite réglées pour l'ATSP et comparées.

De plus, les mêmes opérateurs que pour le QAP sont conservés bien qu'ils ne soient pas réellement adaptés à la résolution du l'ATSP. Ceci teste l'aptitude de la méthode à surmonter des opérateurs mal choisis. Un opérateur supplémentaire spécifique à l'ATSP est ajouté afin d'être en mesure de produire des solutions améliorantes. Le 3-opt [9] $(\sigma_B I, \mathcal{N}_{EE}^2)$ sélectionne la meilleure solution obtenue en supprimant trois côtés et en reconstruisant trois nouveaux de sorte qu'aucun sous-chemin ne soit inversé. $\mathcal{N}_{EE}$ est défini par $(s, s') \in \mathcal{N}_{EE}$ ssi $s = (\pi_1, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j, \pi_{j+1}, \ldots, \pi_n)$ et $s' = (\pi_1, \ldots, \pi_i, \pi_j, \pi_{j-1}, \ldots, \pi_{i+1}, \pi_{j+1}, \ldots, \pi_n)$ pour $i + 1 < j$.

Dans la suite nous examinons la performance de la première combinaison gagnantes par rapport aux résultats précédemment obtenus.

6.3 Analyse des résultats

Les nouveaux résultats pour le QAP sont présentés dans la tableau 2, à la colonne $U_\alpha^{\Sigma+}CD$. Il semble claire que CD est meilleure que les autres méthodes, ou du moins à 0.05 % du meilleur résultats, sur la majorité des instances. Cette supériorité est confirmée par un test de rangs signés de Wilcoxon avec un seuil de confiance de 95 %. Si nous comparons $U_\alpha^{\Sigma+}CD$ aux meilleures valeurs sur les différents α pour $U_\alpha^{\Sigma+}$, les deux distributions sont statistiquement équivalentes. Ceci nous permet de conclure que la stratégie CD est suffisamment performante pour produire des résultats équivalents aux meilleurs résultats de $U_\alpha^{\Sigma+}$ avec un α préréglé.

Les résultats pour l'ATSP sont donnés dans le tableau 3. Les colonnes 4 et 6 utilisent les paramètres choisis pour le QAP. Les colonnes 3, 5 et 7 (astérisques dans les entêtes ds colonnes) utilisent des paramètres réglés pour l'ATSP : $\epsilon = 0.01$ pour U_P avec PR, ($\alpha = 0.3$, $\beta = 0.1$, $p_{min} = 0.001$) pour AP, et ($T_{max} = 0.6$, $T_{min} = 0.25$, $s_{inc} = 0.0001$, $s_{dec} = 0.01$) pour CD.

Si l'on considère les résultats pour le paramétrage spécifique au QAP, les différentes méthodes parviennent à gérer la population d'opérateurs très défavorable. Pour preuve, les résultats pour la sélection uniforme sont très mauvais et rappelons-nous qu'il n'y a qu'un seul opérateur qui soit un bon opérateur pour l'ATSP. D'autre part, AP est peu performante sur les plus grandes instances. Bien entendu, de meilleurs résultats bruts auraient été obtenus si la population d'opérateurs avait été pensée pour l'ATSP. Comme avec le QAP, la stratégie CD produits de bons résultats mais ici elle ne parvient à dépasser les autres méthodes que sur les plus petites instances.

Considérons maintenant les résultats pour le paramétrage spécifique à l'ATSP. Faute de place les résultats pour U_P avec PR ne sont pas inclus mais sont disponibles dans [26]. Par rapport à PR, PR* produit une amélioration sur les plus grandes instances contrebalancée par des résultats de qualité inférieure sur les autres instances. De fait, PR* n'est pas une amélioration en général mais sert plutôt à souligner l'importance du paramètre ϵ. Une valeur plus petite signifie une faible probabilité de sélection des opérateurs très perturbateurs et donc une meilleure performance sur les plus grandes instances. Pour U_P avec AP, AP* donne une amélioration significative. Cette méthode de sélection est très sensible à la configuration de ses paramètres et à la population d'opérateurs. En effet, ici il y a un seul opérateur qui puisse bénéficier de la stratégie du gagnant remporte tout. Néanmoins, les résultats sont équivalents à ceux de la sélection PR* qui est bien plus simple.

Les nouveaux paramètres pour CD ne semblent pas apporter de réel changement positif ou négatif. La faible performance de CD sur les plus grandes instances provient du fait qu'à partir du moment où un fort biais vers l'exploration apparaît, ce biais ne diminue pas suffisamment rapidement, ce qui implique trop d'applications des opérateurs perturbateurs.

La stratégie CD soit basée sur plusieurs paramètres, toutefois les résultats pour CD et CD* montrent que l'algorithme n'est pas hypersensible à leurs valeurs. CD est robuste. Bien que les paramètres demandent un réglage, les résultats tendent à montrer que ce réglage n'est pas forcément à refaire lorsqu'un nouveau problème se présente.

7 Conclusion

Dans cet article nous avons présenté différentes méthodes pour la sélection d'opérateurs en recherche locale. Les opérateurs étaient sélectionnés à partir de valeurs d'utilité calculées en fonction de deux critères : la qualité et la distance par rapport à la trajectoire de

TABLE 2 – QAP – le pourcentage moyen de différence par rapport aux meilleures valeurs connues.

Inst.	Unif.	U_P PR	U_P AP	$U_\alpha^{\Sigma+}$ $\alpha\vert^{0.2}_{0.5}_{0.8}$	$U_\alpha^{\Sigma+}$ CD	RTS
bur26a	*0.03*	**0.00**	*0.02*	*0.02* *0.01* **0.00**	*0.01*	0.00
chr25a	20.2	**10.7**	13.82	13.8 14.9 12.1	12.5	7.1
kra30a	2.49	0.79	1.54	1.57 1.63 0.89	**0.61**	0.06
kra30b	1.11	0.21	0.30	*0.16* 0.45 0.32	**0.13**	0.02
nug20	0.12	*0.01*	*0.01*	**0.00** *0.03* **0.00**	*0.01*	0.00
nug30	1.24	0.20	0.32	0.31 0.19 0.39	**0.11**	0.01
sko42	2.28	0.29	0.38	*0.19* 0.28 0.67	**0.16**	0.03
sko49	2.48	0.36	0.43	**0.21** 0.27 0.81	*0.24*	0.13
tai30a	2.59	1.26	1.50	1.17 1.27 1.68	**0.91**	0.51
tai50a	4.20	2.16	2.13	**1.58** 1.59 2.83	1.66	1.39
tai30b	0.43	*0.13*	0.25	0.44 0.35 *0.16*	*0.15*	0.03
tai50b	2.36	0.25	0.36	0.30 0.39 0.37	**0.18**	0.15
wil50	1.24	0.16	0.15	*0.09* *0.11* 0.28	**0.08**	0.05

TABLE 3 – ATSP – pourcentage moyen de différence par rapport aux meilleures valeurs connues.

Inst.	Unif.	U_P PR*	U_P AP	U_P AP*	$U_\alpha^{\Sigma+}$ CD	$U_\alpha^{\Sigma+}$ CD*
ry48p	14.0	1.5	2.4	1.9	**0.6**	0.7
ft53	32.5	2.0	1.1	2.5	0.9	**0.7**
ft70	10.8	**0.6**	1.3	0.7	0.8	0.7
ftv44	23.8	2.3	2.5	2.6	**0.9**	1.0
ftv47	27.5	1.5	1.7	1.1	**0.5**	**0.5**
ftv55	22.5	2.8	3.5	3.2	**1.2**	1.3
ftv64	32.5	3.5	4.4	3.1	1.7	**1.6**
ftv70	26.9	3.9	5.1	3.5	2.2	**2.0**
ftv90	30.8	6.0	10.7	4.9	**4.0**	4.4
ftv100	32.5	4.8	14.2	5.0	**4.4**	5.3
ftv120	38.4	6.8	21.7	**6.7**	8.6	8.5
ftv140	43.0	**8.3**	29.6	8.4	11.1	12.6
ftv160	42.4	8.3	37.7	**8.2**	15.1	16.6
kro124p	28.0	4.5	8.7	3.6	**3.4**	**3.4**
rbg323	31.7	0.1	12.3	**0.0**	3.3	3.0
rbg403	36.2	**0.0**	3.2	**0.0**	0.3	0.3
rbg443	39.9	**0.0**	4.5	**0.0**	1.0	0.6

recherche. Une contribution de cet article est l'introduction d'une nouvelle utilité pondérée qui autorise un compromis entre exploration et exploitation. En utilisant une pondération statique, cette utilité s'est montrée compétitive par rapport à l'utilité basée sur la Pareto dominance. Une stratégie adaptative pour faire varier la pondération a été étudiée et a offert une amélioration des résultats.

Comme perspectives de recherche, nous souhaitons étudier de nouvelles stratégies adaptative de sélection d'opérateurs. Par exemple, comme cela a été montré sur l'ATSP, la réduction linéaire du poids α n'est pas une stratégie optimale et pourrait être améliorée. Nous souhaitons également augmenter le nombre de problèmes testés et notamment passer aux problèmes booléens. Un autre objectif est la comparaison avec des méthodes de sélection où la probabilité de sélection des opérateurs est réglée hors-ligne, ce qui nécessite un temps de calcul conséquent.

Note. Ces travaux sont financés par Microsoft Research à travers le PhD Scholarship Programme ainsi que par CONICYT–ECOS (Projet C10E07).

Références

[1] R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*, volume 45 of *Operations research/Computer Science Interfaces*. Springer Verlag, 2008.

[2] R. Battiti and G. Tecchiolli. The Reactive Tabu Search. *INFORMS JOURNAL ON COMPUTING*, 6(2) :126–140, January 1994.

[3] M. Birattari. *The Problem of Tuning Metaheuristics as seen from a machine learning perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, December 2004.

[4] R.E. Burkard, S.E. Karisch, and F. Rendl. QAPLIB – A Quadratic Assignment Problem Library. *Journal of Global Optimization*, 10 :391–403, 1997.

[5] E.K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-Heuristics : An Emerging Direction in Modern Search Technology. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Ma-*

nagement Science, pages 457–474. Springer New York, 2003.

[6] P. Codognet and D. Diaz. Yet Another Local Search Method for Constraint Solving. In K. Steinhöfel, editor, *Stochastic Algorithms : Foundations and Applications*, volume 2264 of *Lecture Notes in Computer Science*, pages 342–344. Springer Berlin / Heidelberg, 2001.

[7] A. E. Eiben and S. K. Smit. Evolutionary Algorithm Parameters and Methods to Tune Them. In Hamadi et al. [11], pages 15–36.

[8] Á. Fialho. *Adaptive Operator Selection for Optimization*. PhD thesis, Université Paris-Sud 11, Orsay, France, December 2010.

[9] G. Gutin and A.P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, 2002.

[10] Y. Hamadi, É. Monfroy, and F. Saubion. What Is Autonomous Search ? In P. van Hentenryck and M. Milano, editors, *Hybrid Optimization*, volume 45 of *Springer Optimization and Its Applications*, pages 357–391. Springer New York, 2011.

[11] Y. Hamadi, É. Monfroy, and F. Saubion, editors. *Autonomous Search*. Springer Berlin Heidelberg, 2012.

[12] H.H. Hoos. Automated Algorithm Configuration and Parameter Tuning. In Hamadi et al. [11], pages 37–71.

[13] F. Hutter, H.H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS : an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1) :267–306, September 2009.

[14] L. Ingber. Adaptive Simulated Annealing (ASA) : Lessons learned. *Control and Cybernetics*, 25 :33–54, 1996.

[15] F.G. Lobo, C.F. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

[16] J.B. Mairy, Y. Deville, and P. Van Hentenryck. Reinforced Adaptive Large Neighborhood Search. In *Eighth International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS2011). A Satellite Workshop of CP 2011*, Italy, 2011.

[17] J.B. Mairy, P. Schaus, and Y. Deville. Generic Adaptive Heuristics for Large Neighborhood Search. In *Seventh International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS2010). A Satellite Workshop of CP 2010*, Scotland, 2010.

[18] J. Maturana, Á. Fialho, F. Saubion, M. Schoenauer, and M. Sebag. Extreme compass and Dynamic Multi-Armed Bandits for Adaptive Operator Selection. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 365–372, 2009.

[19] J. Maturana, F. Lardeux, and F. Saubion. Autonomous operator management for evolutionary algorithms. *Journal of Heuristics*, 16 :881–909, 2010.

[20] J. Maturana and F. Saubion. On the Design of Adaptive Control Strategies for Evolutionary Algorithms. In N. Monmarché, E.G. Talbi, P. Collet, M. Schoenauer, and E. Lutton, editors, *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 303–315. Springer Berlin / Heidelberg, 2008.

[21] G. Reinelt. TSPLIB - a traveling salesman problem library. *INFORMS JOURNAL ON COMPUTING*, 3(4) :376–384, January 1991.

[22] É.D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5) :443–455, 1991.

[23] D. Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 1539–1546. ACM, 2005.

[24] D.A.D. Tompkins, A. Balint, and H.H. Hoos. Captain Jack : New Variable Selection Heuristics in Local Search for SAT. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011*, volume 6695, pages 302–316. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[25] N. Veerapen, J. Maturana, and F. Saubion. A Comparison of Operator Utility Measures for Online Operator Selection in Local Search. In *Proceedings of the 6th Learning and Intelligent Optimization Conference (LION6)*, Paris, France, January 2012. À paraître.

[26] N. Veerapen, J. Maturana, and F. Saubion. An Exploration-Exploitation Compromise-Based Adaptive Operator Selection for Local Search. In *Genetic and Evolutionary Computation Conference (GECCO'12)*, Philadelphie, USA, July 2012. À paraître.

[27] N. Veerapen and F. Saubion. Pareto Autonomous Local Search. In C. Coello, editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 392–406. Springer Berlin / Heidelberg, 2011.

Heuristiques de révision et contraintes hétérogènes

Julien Vion

Université de Lille Nord de France, UVHC,
LAMIH CNRS UMR 8201,
59313 Valenciennes Cedex 9, France.
julien.vion@univ-valenciennes.fr

Résumé

La plupart des solveurs de contraintes utilisent une méthode de propagation basée sur l'algorithme générique AC-5 [24]. Le principe d'AC-5 est d'abstraire la notion de *révision de contrainte*. Chaque contrainte dispose d'un *propagateur*, doté d'un algorithme de propagation spécifique, de complexité et performance variables.

Plusieurs travaux ont, par le passé, montré que l'ordre dans lequel les contraintes sont révisées ont un impact non négligeable sur les performances de la propagation [27, 8, 22, 1]. Cependant, la plupart de ces articles se basent sur l'utilisation de propagateurs homogènes, en particulier pour des contraintes binaires définies en extension.

Cet article présente différentes idées et techniques permettant un ordonnancement fin des contraintes au sein d'un système de propagation.

Abstract

Most constraint solvers use the general AC-5 scheme to handle constraint propagation [24]. AC-5 generalizes the concept of *constraint revision*. Each constraint type can thus be shipped with its own revision algorithm, with various complexities and performances.

Previous papers showed that the order in which constraints are revised have a non-negligible impact on performances of propagation [27, 8, 22, 1]. However, most of the ideas presented on these papers are based on the use of homogeneous propagators for binary constraints defined in extension.

This paper give ideas to handle heterogeneous constraints in a general revision schedule.

1 Introduction

La méthode de résolution la plus courante pour résoudre un CSP consiste à effectuer successivement des étapes d'*hypothèse*, de *propagation* et éventuellement de *retour-arrière* (*backtrack*). La propagation consiste généralement à établir la *consistance d'arc*, c'est-à-dire à supprimer toutes les valeurs qui apparaissent inconsistantes (i.e., ne pouvant apparaitre dans aucune solution du CSP) du point de vue d'une contrainte donnée. L'algorithme obtenu, décrit ci-dessus, est appelé `MAC` (*Maintaining Arc-Consistency*). La fonction `AC` (décrite ci-après) réalise la propagation, consistant à supprimer les valeurs inconsistantes du domaine des variables. Elle peut renvoyer la valeur spéciale $\bot$ si le CN est trivialement inconsistant (i.e., le domaine d'une variable est vide, ou une contrainte ne peut plus être satisfaite), ou $\top$ si le CN est trivialement consistant (i.e., toutes les variables ont été réduites à des singletons). L'étape d'hypothèse consiste à effectuer une décision logique, ici notée δ (qui ne doit pas être trivialement impliquée par le CN pour éviter une récursion infinie), consistant le plus souvent à affecter une valeur à une variable non-singleton. Le choix de la variable à affecter peut être réalisé par une heuristique comme $^{dom}/_{ddeg}$ ou $^{dom}/_{wdeg}$ [7].

Fonction MAC($\mathcal{N}$) =

```
1 N' ← AC(N)
2 suivant N' faire
3     cas où ⊥ retourner false
4     cas où ⊤ retourner true
5     autres cas
6         Soit δ une hypothèse logique
7         retourner MAC(N' + δ) ∨ MAC(N' + ¬δ)
```

La suppression de valeurs inconsistantes du point de vue d'une contrainte donnée est appelée *révision de contrainte*. Les algorithmes de propagation comme AC-5 réalisent ces révisions jusqu'à l'obtention d'un point fixe. Dans le cas de l'arc-consistance, ce point

fixe est unique et peut être obtenu en un temps fini. Cependant, établir l'arc-consistance est NP-difficile dans le cas général (i.e., si l'on n'a aucune information sur la nature des contraintes). Pour cette raison, pendant longtemps, seuls les CSP n'impliquant que des contraintes binaires (exactement deux variables par contrainte) étaient traités. Le CSP binaire est aussi NP-complet, mais on peut établir l'arc-consistance sur un CN binaire en temps et en espace polynomiaux. Les contraintes binaires restent malheureusement bien moins expressives que les contraintes non-binaires [4].

Des dizaines d'algorithmes ont été proposés durant les 40 dernières années pour effectuer la phase de propagation : AC-1 à AC-8, plusieurs variantes d'AC-3, etc. La plupart de ces algorithmes peuvent être décomposés en un algorithme de *propagation* et un algorithme de *révision de contrainte* (NP-difficile ou limité aux contraintes binaires). Cependant, deux algorithmes, AC-5 (1992 [24]) et GAC-schema (1997 [5]), traitent la révision de contrainte de manière *abstraite* : l'idée générale d'AC-5 est de considérer que chaque contrainte du CN dispose de propriétés sémantiques pouvant être exploitées pour développer des algorithmes efficaces de révision de contraintes.

La plupart des solveurs de CSP sont basés sur AC-5, et proposent une « boite à outils » de contraintes usuelles, disposant chacune de leur propagateur [2]. Certaines de ces contraintes, définies uniquement à partir de leur propriété sémantique et pouvant impliquer un nombre arbitraire de variables sont appelées *contraintes globales* [3].

Tous les algorithmes d'arc-consistance depuis AC-3 [18], sont basés sur une *file de propagation*. Quand une valeur est supprimée du domaine d'une variable, toutes les contraintes impliquant cette variable peuvent avoir perdu leur propriété d'arc-consistance, et doivent être révisées afin de propager la valeur supprimée aux autres variables du réseau. La nature des données contenues dans la file de propagation (variables, contraintes et/ou valeurs), ainsi que l'ordre dans lequel les révisions sont effectuées ont un impact significatif sur les performances de la propagation. Plusieurs travaux ont cherché à déterminer un « bon » ordonnancement des révisions [27, 8, 1], mais se sont souvent limités aux CSP binaires et propagateurs généraux. Appliquer ces techniques de propagation à des propagateurs hétérogènes peut entrainer des comportement pathologiques : typiquement, un propagateur très lent ou très faible devrait être exécuté le moins souvent possible (on trouvera un exemple au début de la section 4.4).

Les contributions de cet article sont les suivantes :

1. établir un état de l'art d'algorithmes de propagation « à gros grain » supportant la notion de propagateur abstrait (section 3) ;
2. évaluer des structures de données permettant de gérer la file de propagation (section 4.2) ;
3. abstraire la notion d'heuristique d'ordonnancement des révisions au niveau des contraintes pour obtenir un ordre de priorité fin et adapté au propagateur de la contrainte (section 4.4).

2 Définitions et notations

Définition 1 (Réseau de contraintes, variable, domaine, contrainte)**.** Un *réseau de contraintes* $\mathcal{N}$ est un couple $(\mathscr{X}, \mathscr{C})$ constitué :

- d'un ensemble de n *variables* $\mathscr{X}$; un *domaine* $\mathrm{dom}(X)$ est associé à chaque variable $X \in \mathscr{X}$ et définit l'ensemble fini d'au plus d valeurs auxquelles X peut être *instanciée*, et
- d'un ensemble de e *contraintes* $\mathscr{C}$; chaque contrainte $C \in \mathscr{C}$ implique au plus k variables $\mathrm{vars}(C) \subseteq \mathscr{X}$; la contrainte définit les combinaisons de valeurs auxquelles ces variables peuvent être instanciées.

On appelle une combinaison de valeurs affectées à un ensemble de variables une *instanciation*. L'ensemble des contraintes impliquant une variable donnée X est noté $\mathrm{ctr}(X)$. Une contrainte peut être définie en *extension*, c'est-à-dire par une liste exhaustive d'instanciations autorisées ou interdites, ou en *intention*, c'est-à-dire par une fonction : $\prod_{X \in \mathrm{vars}(C)} \mathrm{dom}(X) \rightarrow \mathbb{B}$. Les contraintes dites *globales* définissent une propriété d'arité arbitraire que les variables qu'elle implique doivent vérifier (par exemple, *all different*).

Le problème de satisfaction de contraintes (CSP) consiste à décider si une solution au CN (c'est-à-dire une instanciation de toutes les variables satisfaisant toutes les contraintes du CN) existe. Quand toutes les contraintes peuvent être vérifiées en temps et en espace polynomiaux, le CSP est NP-complet.

Définition 2 (Arc-consistance, support)**.** Soit C une contrainte et $X \in \mathrm{vars}(C)$. La valeur $v \in \mathrm{dom}(X)$ est arc-consistante (AC) pour C ssi il existe une instanciation de $\mathrm{vars}(C)$, autorisée par C, qui instancie X à v (une telle instanciation est appelée un *support* de v pour C). C est AC ssi $\forall X \in \mathrm{vars}(C)$, $\forall v \in \mathrm{dom}(X)$, v est AC pour C.

Dans la littérature, la définition de l'arc-consistance est souvent restreinte aux CN binaires, et l'extension d'AC aux CN non-binaires est appelée AC généralisée (GAC), Hyper-AC ou consistance de domaine (*Domain Consistency*). Dans cet article, nous nous référons à AC pour les CN binaires et non binaires.

Définition 3 (Clôture)**.** Soit $\mathcal{N} = (\mathscr{X}, \mathscr{C})$ un réseau de contraintes. $\text{AC}(\mathcal{N}, C)$ est la *clôture* de $\mathcal{N}$ par AC sur C, c'est-à-dire le CN obtenu à partir de $\mathcal{N}$ tel que $\forall X \in \text{vars}(C)$, toutes les valeurs $v \in \text{dom}(X)$ qui ne sont pas AC pour C ont été supprimées.

$\text{AC}(\mathcal{N})$ est la clôture de $\mathcal{N}$ par AC, c'est-à-dire le CN obtenu à partir de $\mathcal{N}$ tel que $\forall C \in \mathscr{C}$, C a été rendu AC par clôture.

Pour tout $\mathcal{N} = (\mathscr{X}, \mathscr{C})$, pour tout $C \in \mathscr{C}$, $\text{AC}(\mathcal{N}, C)$ et $\text{AC}(\mathcal{N})$ sont uniques. Dans le cas général, calculer la clôture par AC d'un CN est NP-difficile. L'algorithme optimal GAC-schema admet une complexité en $O(ekd^k)$ [5].

Définition 4 (Propagateur)**.** Étant donné un CN $\mathcal{N} = (\mathscr{X}, \mathscr{C})$, le *propagateur* d'une contrainte $C \in \mathscr{C}$ est l'algorithme permettant d'obtenir $\text{AC}(\mathcal{N}, C)$.

3 Algorithmes de propagation

Les algorithmes présentés dans cette section, inspirés d'AC-3 et AC-5, n'ont pas pour vocation d'être innovants, mais plutôt de présenter un état de l'art des algorithmes de propagation à gros grain. La plupart des solveurs disponibles (Choco [11], Gecode [21], Abscon [20], CSP4J [25], Eclipse…) fournissent une variante des algorithmes présentés ici, mais en proposant généralement un grain fin.

3.1 Propagateurs abstraits

Historiquement, la plupart des algorithmes d'AC ont été conçus en considérant que la seule information disponible pour une contrainte était le résultat de sa fonction de vérification (*check*). Dans ces conditions, la seule manière de d'obtenir l'arc-consistance consiste à trouver un support de chaque valeur pour chaque contrainte du CN, en énumérant toutes les instanciations possibles des variables impliquées par chaque contrainte, d'où une complexité minimale en $O(ekd^k)$ pour établir AC (en considérant un *check* en $O(k)$). AC-5 et GAC-schema proposent des hypothèses plus générales, aboutissant au total à trois niveaux d'abstraction, du plus bas au plus haut niveau :

- *Vérification de contraintes*, l'hypothèse historique ;
- *Recherche de support*, proposée par GAC-schema : chaque contrainte dispose d'un service permettant de trouver rapidement un support d'une valeur ;
- *Révision de contraintes*, proposée par AC-5.

En réalité, l'hypothèse de *recherche de support abstraite* peut s'appliquer à tous les algorithmes d'AC (sauf AC-5) sans que ceux-ci ne perdent leur spécificité. C'est dans la manière de mémoriser les supports pour obtenir des propriétés d'incrémentalité que ces algorithmes diffèrent. On peut ainsi extraire un propagateur *général* de chacun de ces algorithmes. Un exemple est donné ci-après avec le propagateur `revise`$^{\texttt{rm}}$.

3.2 Files de propagation

Indépendamment du propagateur général, les algorithmes d'AC sont généralement classés en deux familles, en fonction de la nature des données stockées dans la file de propagation. Les algorithmes dits « à grain fin » (AC-4 à AC-7 et GAC-schema) stockent chaque *valeur* récemment supprimée dans la file de propagation, et cherchent à exploiter cette information pour éviter des calculs inutiles. Les algorithmes « à gros grain » (variantes d'AC-3 et AC-8) stockent les variables modifiées (i.e., une valeur a été supprimée de leur domaine) et/ou les contraintes impliquant celles-ci, qui doivent donc être révisées. Bien que le grain fin soit indispensable pour concevoir des algorithmes optimaux, la différence théorique reste marginale (l'algorithme à gros grain GAC-2001 admet une complexité en $O(ek^2d^k)$ [6]), et les structures de données plus simples permettent souvent d'obtenir de meilleurs résultats en pratique. Dans la suite de cette section, nous décrivons des techniques permettant de limiter le plus possible les révisions inutiles dans ce cadre des files à gros grain.

L'algorithme AC-3 de Mackworth (1977 [18]) était un algorithme à gros grain « orienté arcs », c'est-à-dire qu'il utilisait une file de propagation contenant des *arcs* constitués de couples (*Variable*, *Contrainte*). La *Variable* du couple identifie une variable dont le domaine peut ne pas être AC pour la *Contrainte*. L'algorithme de révision doit alors chercher un support pour chaque valeur de la *Variable* pour la *Contrainte*. McGregor montre que le même comportement peut être obtenu en ne stockant que les variables modifiées dans la file (1979 [19]). Cependant, si l'on travaille avec des contraintes non-binaires, une telle file ne contient pas suffisamment d'information pour éviter autant de révisions inutiles : si deux variables impliquées par la même contrainte non-binaire se trouvent dans la file, le domaine de chacune de ces variables ne devrait être contrôlé qu'une seule fois, ce qu'un algorithme orienté variables ne permet pas de faire.

Boussemart *et al.* ont proposé en 2004 une version de l'algorithme basé sur une file de variables, disposant d'une structure de données complémentaire que nous appelons *modif* afin d'émuler les avantages de la file à base d'arcs pour une propagation orientée variables [8]. Cette structure de données associe à chaque contrainte la liste des variables modifiées depuis la dernière révision de celle-ci. *modif* permet de connaitre les arcs à réviser et contient donc les mêmes informa-

Algorithme 1: updateModif($modif, \Delta, C$)

```
1 modif(C) ← ∅
2 pour chaque Y ∈ Δ, C' ∈ ctr(Y) − C faire
3     modif(C') ← modif(C') ∪ {Y}
```

Algorithme 2: AC-V($\mathcal{N}, Q, modif$) =

```
1 si Q = ∅ alors retourner N
2 sinon
3     Choisir X dans Q, Q' ← Q − X
4     pour chaque C ∈ ctr(X) | modif(C) ≠ ∅
      faire
5         (N', Δ) ← revise(C, modif(C))
6         si Δ = ⊥ alors retourner ⊥
7         updateModif(modif, Δ, C)
8         Q' ← Q' ∪ Δ
9     retourner AC-V(N', Q', modif)
```

tions qu'une file de propagation orientée arcs. Cependant, regrouper les révisions par contrainte simplifie considérablement la conception des propagateurs, particulièrement si l'on fait l'hypothèse de propagateurs abstraits. On trouve aussi des mécanismes plus sophistiqués, par exemple dans le solveur Gecode [21, 12] : l'idée des *advisors* est d'abstraire, en plus du propagateur, la détection des révisions inutiles.

L'algorithme 1 montre comment la structure est mise à jour, après qu'une contrainte C ait supprimé des valeurs dans les variables Δ. C n'a plus besoin d'être révisée pour l'instant (ligne 1), puis, pour chaque contrainte C' impliquant une variable de Δ, on indique que la variable correspondante a été modifiée (en prenant soin de ne pas altérer $modif(C)$).

Cette structure de données peut également être utilisée pour concevoir un algorithme de propagation efficace basé uniquement sur une file de contraintes. Nous décrivons donc deux algorithmes qui nous serviront de base pour nos développements, l'un centré sur une file de variables, l'autre sur une file de contraintes. Nous conservons l'idée d'abstraire la notion de révision de contraintes, comme dans l'algorithme AC-5, mais avec des files à gros grain. La gestion de la structure *modif* présentée ici améliore légèrement les versions de Boussemart *et al.* : au maximum un surcoût en $O(k)$ dans l'algorithme 4 (ligne 3) contre $O(k^2)$ dans la version originale.

3.3 Propagation orientée variables

Dans l'algorithme AC-V (algorithme 2), la file de propagation Q contient la liste des variables modifiées, qui nécessitent une révision des contraintes les

Algorithme 3: AC-C($\mathcal{N}, Q, modif$) =

```
1 si Q = ∅ alors retourner N
2 sinon
3     Choisir C dans Q, Q' ← Q − C
4     (N', Δ) ← revise(C, modif(C))
5     si Δ = ⊥ alors retourner ⊥
6     sinon
7         updateModif(modif, Δ, C)
8         Q' ← Q' ∪ {C' | vars(C') ∩ Δ ≠ ∅}
9         retourner AC-C(N', Q', modif)
```

impliquant. Q et *modif* doivent être initialisés conformément à l'état connu de $\mathcal{N}$ (par exemple, au sein de `MAC` on pourra initialiser Q et *modif* avec les variables impliquées par les hypothèses δ).

À chaque itération de l'algorithme, une variable est sélectionnée (ligne 3), et toutes les contraintes impliquant cette variable sont traitées (boucle de la ligne 4). Notez que plusieurs variables impliquées par une même contrainte peuvent se trouver simultanément dans la file. Le test $modif(C) \neq \emptyset$ permet d'éviter de propager inutilement à chaque fois les mêmes contraintes.

La ligne 5 fait appel au propagateur spécifique de C. Le propagateur renvoie le réseau filtré ainsi que l'ensemble $\Delta \subseteq \text{vars}(C)$ des variables modifiées, ou $\perp$ si une inconsistance a été détectée (par exemple, le domaine d'une variable a été vidé).

3.4 Propagation orientée contraintes

Dans cette version, appelée AC-C (algorithme 3), ce sont les contraintes à réviser qui sont stockées dans la file. L'algorithme obtenu est légèrement plus simple, puisque l'on n'a pas à gérer le cas des multiples révisions d'une même contrainte. La structure *modif* sera exploitée uniquement par les propagateurs.

3.5 Le propagateur général `revise`$^{\text{rm}}$

Pour illustrer l'utilisation du système de propagation spécialisable, nous montrons un exemple de propagateur nommé `revise`$^{\text{rm}}$. Il s'agit d'un propagateur issu de l'algorithme AC-3$^{\text{rm}}$ [15]. À l'image de GAC-schema, nous avons abstrait la notion de recherche de support (fonction abstraite `findSupport`). L'idée est de rechercher un support pour chaque valeur, et d'enregistrer ce support dans la structure *res*. Ainsi, la prochaine fois que l'on cherchera un support pour la même valeur, on commence par vérifier si le support enregistré (nommé *résidu*) est toujours valide.

Notez l'exploitation de la structure *modif* à la ligne 3 : si une seule variable a été modifiée, les au-

Algorithme 4: revise$^{\text{rm}}(C, \mathit{modif})$ =

```
1  N' ← N
2  Δ ← ∅
3  pour chaque X ∈ vars(C) | modif ≠ {X},
     v ∈ dom(X) | res(C, X, v) non valide faire
4      τ ← findSupport(C, X, v)
5      si τ = ⊥ alors
6          Supprimer v de dom(X) dans N'
7          si dom(X) = ∅ alors retourner ⊥
8          Δ ← Δ ∪ {X}
9      sinon pour chaque Y ∈ vars(C) faire
10         res(C, Y, τ(Y)) ← τ
11 retourner (N', Δ)
```

ters valeurs de la variable n'ont pas pu perdre leurs supports et il est inutile de les parcourir.

4 Gestion de la file de propagation

Note : Toutes les heuristiques présentées dans cet article s'entendent « valeur minimale d'abord ».

4.1 Travaux connexes sur les heuristiques d'ordonnancement des révisions

L'ordre dans lequel les contraintes sont révisées a un impact non négligeable sur les performances de la propagation, et plusieurs travaux ont été consacrés à étudier un « bon » ordre de propagation de ces contraintes. Le premier article sur le sujet est de Wallace & Freuder (1992 [27]). Leur étude compare l'impact de différentes heuristiques d'ordonnancement sur un algorithme de propagation AC-3 orienté arcs sur des CSP binaires. Les heuristiques conçues par Wallace & Freuder suivent le principe suivant : pour une propagation efficace, il faut supprimer des valeurs le plus tôt possible. Cependant, il est très difficile de prévoir si une contrainte est susceptible de supprimer des valeurs avant sa propagation.

Wallace & Freuder utilisent la dureté (proportion d'instanciations des domaines initiaux des variables interdites par la contrainte) pour estimer la force de celle-ci. Calculer la dureté d'une contrainte reste #P-difficile, et la dureté de celle-ci peut fortement évoluer au cours de la recherche, lorsque des valeurs sont supprimées des domaines des variables. Cette heuristique est donc inapplicable dynamiquement ou dans un cadre non-binaire. D'autres heuristiques moins complexes sont proposées : la plus efficace évalue uniquement la taille du domaine de la variable dans l'arc. Cependant, même sur de petits CSP binaires, les meilleurs résultats de Wallace & Freuder sont obtenus sur des heuristiques dites statiques, calculées une seule fois au début de la recherche. Une alternative intéressante, plus récemment proposée par Balafoutis & Stergiou (2010 [1]), est d'exploiter les poids des contraintes calculés par l'heuristique d'instanciation $^{dom}/_{wdeg}$ [7] pour estimer les contraintes les plus fortes sans surcout. Cette heuristique n'est évaluée que sur des CSP binaires homogènes.

Boussemart *et al.* ont étudié et évalué (2004 [8]) différentes heuristiques d'ordonnancement des révisions en utilisant des algorithmes de propagation orientés arcs, variables ou contraintes. Comme dans les travaux précédemment cités, ces évaluations se restreignent à des CSP binaires et homogènes. Le principal résultat de cette étude est que, dans ce contexte, la variante la plus efficace est un algorithme de propagation orienté variables, en propageant d'abord les modifications des variables dont le domaine est le plus petit (nous appelons cette heuristique *dom*). Ce résultat est très proche des conclusions de Wallace & Freuder (à ceci près que Wallace & Freuder considèrent la taille du domaine de la variable *à réviser*). On note qu'une propagation orientée contraintes (ou arcs), associée à une heuristique consistant à réviser en premier la contrainte dont le produit de la taille des domaine des variables impliquées est le plus petit (que nous appelons Πdom) parvient à limiter le nombre de révisions par rapport à l'approche orientée variables, mais le surcout représenté par le calcul de l'heuristique est trop important. Cependant, les structures de données employées par Boussemart *et al.* (cf ci-après) peuvent être considérablement améliorées.

Le solveur Gecode utilise une méthode de propagation relativement sophistiquée [22]. Ici, la stratégie diffère de celle proposée initialement par Wallace & Freuder : comme on ne peut pas facilement prévoir si une contrainte va filtrer des valeurs ou non, on cherche à minimiser le temps perdu en propageant les contraintes les plus rapides en premier. La technique utilisée ici ne peut être utilisée qu'avec une propagation orientée contraintes ou arcs. Un entier est associé à chaque contrainte : 1 pour des propagateurs à temps constant, 2 à 4 pour des propagateurs en $O(kd)$ (suivant la valeur de k), 5 et 6, respectivement, pour des propagateurs en $O(kd^2)$ et $O(kd^3)$, ou 7 pour les propagateurs plus lents [1]. L'entier est amené à évoluer en fonction de l'état de la recherche : par exemple, une contrainte ternaire dont une variable a été instanciée peut être considérée comme une contrainte binaire. Chaque entier identifie une file de propagation de type

1. En réalité, les nombres utilisés par Gecode vont de 7 pour les propagateurs les plus rapides à 1 pour les plus lents ; la notation est inversée ici pour correspondre au comportement des autres heuristiques.

Structure	Insert	M. à j.	Extr. min
Bit vector	$O(1)$	$O(1)$	$\Theta(\lambda)$
Binary heap	$O(\log \lambda)$	$O(\log \lambda)$	$O(\log \lambda)$
Binomial h. [26]	$O(1)$*	$O(\log \lambda)$	$O(\log \lambda)$
Fibonacci h. [9]	$O(1)$	$O(1)$*	$O(\log \lambda)$*

TABLE 1 – Structures de données pour files de priorité. λ est le nombre d'éléments présents dans la structure, * indique une complexité amortie.

FIFO ; celles-ci sont traitées par ordre de priorité (voir section suivante). Cette approche est moins fine que les précédentes, mais minimise la surcharge représentée par le calcul des heuristiques. Notons qu'il n'est plus possible de donner une priorité différente à deux contraintes de même type, même si l'une porte sur des domaines bien plus grands que l'autre. Gecode permet cependant de définir manuellement l'ordre de propagation de certaines contraintes pour gérer au mieux certains cas spécifiques [13]. Les solveurs Choco [11] et Minion [10] proposent une stratégie hybride : une file (ou pile) de variables d'abord, tout en reportant la propagation de certaines contraintes plus lentes (identifiées de manière statique, correspondant au niveau 3 ou 4 et plus de Gecode) dans une file à part.

On peut finalement constater que les heuristiques les plus efficaces d'après les articles de Wallace & Freuder ou Boussemart *et al.* (dom et Πdom) se révèlent également suivre le pricipe du « propagateur le plus rapide d'abord » : les algorithmes de propagation utilisés, extraits d'AC-3 pour des contraintes binaires, sont en $O(d^2)$. Choisir les domaines les plus petits en priorité aura donc tendance à sélectionner les propagateurs les plus rapides. Dans le cas de propagateurs hétérogènes, la situation est plus compliquée.

4.2 Structures de données pour les files de priorité

Que l'on utilise une méthode de propagation orientée variables ou contraintes, utiliser une heuristique d'ordonnancement dynamique des révisions nécessite l'utilisation de *files de priorité*. Ces structures supportent trois opérations : insertion d'un élément dans la structure, extraction du plus petit élément, et mise à jour de la valeur d'un élément. Il s'agit d'une problématique abondamment étudiée dans la littérature algorithmique et disposant de centaines d'applications. Le plus souvent, les files de priorité sont des *tas* (*heaps*), c'est-à-dire des structures de données arborescentes équilibrées, construites de sorte que la valeur d'un nœud de l'arbre soit toujours inférieure à celles de ses fils. Les arbres entrainent une complexité logarithmique pour la plupart des opérations.

Il existe de nombreuses variantes de tas, généralement capables d'obtenir des complexités amorties pratiquement constantes. La table 1 montre différentes structures que nous avons implanté et expérimenté dans le cadre de cette étude. La structure de données « Bit vector » est proche de celle utilisé par Boussemart *et al.* Les complexités sont indicatives : une structure sophistiquée entraine souvent un facteur constant plus important qu'une structure plus simple, et les complexités amorties cachent parfois des comportements pathologiques. La section expérimentale de cet article contient une étude empirique du comportement de ces structures de données pour la résolution de CSP.

Une autre approche possible pour la gestion des files de priorité est celle retenue par les développeurs de Gecode : plutôt que d'utiliser un tas, on restreint le nombre de priorités possibles à m valeurs (7 pour Gecode). La file de priorité est divisée en m files traitées successivement. La complexité pour ajouter ou supprimer un élément est alors constante, et la complexité pour extraire le plus petit élément est de $O(m)$. Des pointeurs sont maintenus sur la première et la dernière file non vides, afin d'éviter de parcourir inutilement certaines files inutilisées.

Il est finalement possible d'utiliser cette structure de données pour appliquer une approximation d'une heuristique quelconque : le logarithme de la fonction d'évaluation peut être utilisé pour choisir une des m files de priorité (logarithme à base 16 pour passer d'un nombre codé sur 31 bits à 3 bits). Ce logarithme peut être calculé efficacement en observant la position du bit le plus fort dans la représentation binaire d'un nombre. Ces approches seront comparées aux tas dans la section expérimentale de cet article.

4.3 Les faiblesses de la propagation orientée variables

Soit $\mathcal{N}$ un CN constitué de n variables X_1 à X_n avec $\text{dom}(X_i) = \{1, \ldots, n\}$. Les contraintes sont $X_i \leq X_{i+1}$ et alldifferent$(X_1, \ldots, X_n)$. Le CSP correspondant à $\mathcal{N}$ n'admet qu'une seule solution : $(X_1, \ldots, X_n) = (1, \ldots, n)$. Bien que le problème paraisse trivial (en instanciant $X_1 = 1$, on obtient immédiatement la solution par propagation), une stratégie de résolution classique (en utilisant par exemple l'heuristique d'instanciation $^{dom}/_{ddeg}$) aura peu de chance d'effectuer cette affectation correcte avant de nombreuses réfutations. Pire, une propagation orientée variables adopte un comportement pathologique : après une réfutation, la valeur supprimée est propagée, variable par variable, grâce aux contraintes $\leq$. Cependant, en utilisant une propagation orientée variable, la contrainte alldifferent est inutilement appelée après chaque modification de variable ! En utilisant une propagation orientée con-

Contrainte	Algo. de propag.	Complexité	Évaluateur
$X\{<,\leq\}Y$	Arith. intervalles	$O(1)$	2
$\pm X + c = Y$ (intervalles)	Arith. intervalles	$O(1)$	3
$X = Y\{+,\lvert-\rvert\}Z$ (intervalles)	Arith. intervalles	$O(1)$	4
$\bigvee(\dots)$	*Watched literals*	$O(k)$	$\log_2 \lvert C\rvert$
alldifferent$(\dots)$	L.-O. *et al.* [17]	$O(k \log k)$	$\lvert C\rvert \log_2 \lvert C\rvert$
$X \neq Y$	Algo. bit-à-bit	$O(d)$	$\min(\lvert X\rvert, \lvert Y\rvert)$
$\pm X + c = Y$	Algo. naïf	$O(d)$	$\lvert X\rvert + \lvert Y\rvert$
$X = Y\{+,\times,\lvert-\rvert\}Z$	revise$^{\text{rm}}$ [15]	$O(d^2)$	$\lvert X\rvert\lvert Y\rvert + \lvert X\rvert\lvert Z\rvert + \lvert Y\rvert\lvert Z\rvert$
relations binaires	revise$^{\text{bit+rm}}$ [16]	$O(d^2)$	$\lvert X\rvert\lvert Y\rvert/3$
liste de λ supports	STR2 [14]	$O(k\lambda)$	$\lambda \lvert C\rvert$
liste conflits	revise$^{\text{rm}}$	$O(kd^k)$	$\lvert C\rvert \prod_{X\in\text{vars}(C)} \lvert X\rvert$
$X \iff C(\dots)$	Algo. naïf	Complexité $C + \neg C$	$eval(C) + eval(\neg C)$

TABLE 2 – Propagateurs et évaluateurs des contraintes de CSP4J. $|C|$ représente l'arité de C, $|X|$ est $|\text{dom}(X)|$. Certains propagateurs sélectionnent dynamiquement un algorithme de propagation basé sur l'arithmétique des intervalles quand le domaine des variables s'y prête. Certains propagateurs incluent une détection limitée de contrainte *entailed*.

traintes et en donnant la priorité aux contraintes $\leq$, plus rapides à propager et plus fortes, le propagateur de *alldifferent* n'est appelé qu'une seule fois, ce qui entraine une propagation bien plus rapide. Pour donner une idée de l'impact de la méthode de propagation, notre solveur CSP4J sur notre machine de test (cf section expérimentale ci-après) propage la réfutation $X_2 \neq 1$ en environ 6,5 s pour $n = 2\,000$ avec AC-V contre 0,1 s avec AC-C. Ce problème sera appelé *bigleq-n* dans la section expérimentale de l'article.

Un autre exemple classique est le problème des n-*dames*, modélisé sous la forme des trois contraintes alldifferent$(X_1, \dots, X_n)$, alldifferent$(X_1 - 1, \dots, X_n - n)$ et alldifferent$(X_1 + 1, \dots, X_n + n)$. L'implantation de ce modèle nécessite l'introduction de variables auxiliaires pour représenter les variables $X_i \pm i$, reliées aux variables principales par des contraintes d'égalité (à une constante près). Ces contraintes devraient être propagées d'abord. Ainsi, MAC-$^{dom}/_{ddeg}$ avec AC-V trouve la première solution au problème des 40-dames en 86 s, contre 41 s pour AC-C.

4.4 Heuristiques d'ordonnancement des révisions fines et hétérogènes

Pour déterminer la priorité des contraintes, nous définissons une heuristique comme suit : un *évaluateur* dynamique est associé à chaque contrainte. Un évaluateur est une fonction qui renvoie un nombre entier positif défini sur 31 bits (0 à $2 \cdot 10^9$ environ). L'objectif de cet évaluateur est de pouvoir émuler à la fois les heuristiques proposées par Wallace & Freuder, Boussemart *et al.* ou Balafoutis & Stergiou (dans ce cas, l'évaluateur sera le même pour chaque contrainte), ainsi que de définir une priorité contrainte par contrainte, à la manière de Gecode, mais de manière plus fine et plus souple. Nous avons défini une série d'évaluateurs pour les différentes contraintes implantées dans notre solveur CSP4J ; ces évaluateurs définissent la priorité des contraintes en fonction de leur temps estimé de propagation. La table 2 répertorie ces évaluateurs. Dans la suite de cet article, nous appellerons *eval* l'heuristique d'ordonnancement des révisions utilisant les évaluateurs définis dans cette table.

On peut finalement combiner n'importe quelle heuristique avec le principe développé par Balafoutis & Stergiou [1] : en divisant le résultat donné par l'évaluateur par le poids de la contrainte, on obtient alors les heuristiques $^{eval}/_{w}$ et $^{\Pi dom}/_{w}$.

5 Expérimentations

Ces expérimentations sont réalisées à l'aide de notre solveur CSP4J [25]. Celui-ci a été exécuté sur une machine virtuelle Java OpenJDK IcedTea7 2.1 Server 64 bits pour Linux, fonctionnant sur un processeur Intel Core 2 Quad Q6700. L'heuristique d'instanciation utilisée est $^{dom}/_{ddeg}$ avec résolution pseudo-aléatoire des égalités (la graine est fixe) et redémarrages à progression géométrique. L'heuristique $^{dom}/_{wdeg}$ serait bien plus performante, cependant elle interagit avec l'heuristique de révision et les résultats ne seraient pas comparables. Avec la stratégie de résolution choisie, l'arbre de recherche parcouru par `MAC` est rigoureusement le même quelle que soit la méthode de propagation utilisée. Les différents algorithmes de révision utilisés sont ceux indiqués sur la table 2. Nous avons sélectionné une série d'instances de CSP représenta-

			AC-V/*dom*				AC-C/*eval*			
	n	e	Bit v	B$^{\text{ary}}$ h	B$^{\text{ial}}$ h	Fib. h	Bit v	B$^{\text{ary}}$ h	B$^{\text{ial}}$ h	Fib. h
bigleq-400	400	400	731,5	666,8	702,3	712,4	**19,7**	**18,9**	**18,4**	22,0
bqwh-18-141-0-ext	141	879	37,4	37,4	37,6	38,7	42,5	**31,5**	**33,4**	37,3
bqwh-18-141-47-glb	141	36	**93,9**	**90,3**	**90,9**	**93,3**	**88,0**	**85,6**	**85,4**	**87,1**
crossword-lex-vg5-7	35	12	123,9	124,8	123,9	125,1	**84,3**	**82,9**	**81,3**	**82,9**
frb40-19-2-ext	40	410	45,0	45,0	45,1	45,5	53,4	**35,7**	**37,3**	45,0
js-e0ddr1	1 480	1 205	**305,2**	**280,3**	**289,2**	**304,8**	730,1	368,5	358,7	434,0
langford-3-13	65	27	101,5	104,4	101,4	102,1	**41,3**	**42,6**	**41,9**	**42,1**
lemma-24-3	552	924	70,4	70,1	69,9	72,0	65,1	**58,5**	**60,4**	**62,6**
os-taillard-5-100-4	625	500	**489,9**	**473,3**	**459,5**	**491,7**	701,3	**452,4**	**449,5**	**493,2**
queens-40	120	83	88,9	84,7	83,2	86,7	42,6	**38,2**	**38,9**	**38,7**
ruler-44-9-a3	45	38	17,3	17,6	18,4	17,7	**7,7**	**7,3**	**7,3**	**7,9**
scen11	8 546	7 866	662,7	653,3	657,8	661,1	**616,4**	**576,2**	**587,8**	**590,5**
series-20	77	40	14,8	14,5	15,1	14,9	**9,0**	**9,2**	**9,1**	**9,5**
tsp-20-193-ext	61	230	106,1	112,8	110,0	109,8	**92,4**	**89,9**	**89,3**	**97,0**

TABLE 3 – Performances des différentes files de priorité. Temps de résolution en secondes avec l'heuristique d'instanciation $^{dom}/_{ddeg}$. Les meilleurs temps (dans une marge de 10 %) sont surlignés.

tives, utilisées lors de la compétition internationale de solveurs CPAI'08 [23]. Les instances sont choisies pour être de difficulté moyenne (pouvant être résolue en 10 à 1 000 secondes).

Une première série d'expérimentations, dont les résultats sont indiqués dans la table 3, concerne le choix de la structure de données pour les files de priorité. Les résultats mettent en avant les tas binaires et binomiaux, sans qu'il soit réellement possible de les discriminer, que ce soit pour une propagation orientée variables ou contraintes.

Finalement, la table 4 montre une comparaison des différentes heuristiques de révision. Pour les heuristiques $^{dom}/_{wdeg}$, $^{\Pi dom}/_{w}$ et $^{eval}/_{w}$, on incrémente le poids d'une contrainte quand elle détecte une inconsistance au cours de la propagation, bien que l'heuristique d'instanciation $^{dom}/_{ddeg}$ soit utilisée. En plus des heuristiques présentées précédemment, nous expérimentons de simples files FIFO (réalisées à l'aide de deux listes simplement chainées) pour les variables et les contraintes. La colonne AC-C/8 FIFO/Arity correspond au fonctionnement du solveur Gecode (avec les évaluateurs simplifiés basés sur l'arité de la complexité du propagateur). La dernière colonne est l'hybride entre l'heuristique *eval* et la structure de données composée de 8 files FIFO décrite à la fin de la section 4.2. C'est cette dernière version qui apparait comme étant la plus robuste : elle permet à la fois de discriminer des contraintes de manière plus fine que l'approche Gecode, ce qui permet un gain sensible sur certains problèmes (par exemple, quand ils utilisent des contraintes en extension comme les *frb*, *bqwh-...-ext*, *crossword*), et à la fois de présenter un surcoût minimal pour les problèmes pour lesquels l'ordre des révisions n'a que peu d'impact (par exemple, les problèmes dont les domaines sont toujours des intervalles, comme les problème *js-...* ou *os-...*). En tout état de cause, les propagations orientées variables ne sont jamais parmi les plus performantes.

Les heuristiques prenant en compte le poids des contraintes semblent également peu performantes, mais il convient de rappeler que celles-ci avaient été conçues pour interagir positivement avec l'heuristique d'instanciation $^{dom}/_{wdeg}$, qui n'a pas été utilisée ici.

6 Conclusion & perspectives

Dans cet article, nous avons décrit AC-V et AC-C, deux algorithmes de propagation de contraintes à gros grain spécialisables et utilisant respectivement une file de propagation orientée variables et contraintes. Nous avons rappelé en quoi la gestion de la file de propagation est importante lors de la conception d'un algorithme de propagation, nous avons évalué plusieurs structures de données pouvant être utilisées pour ce faire. Malgré des performances correctes pour les *tas binaires* ou *binomiaux*, la meilleure alternative pour une majorité des problèmes testés s'est révélée être l'utilisation de plusieurs FIFO de priorité croissante.

Après avoir montré comment une propagation orientée variables peut entrainer un comportement pathologique, nous avons proposé une nouvelle manière de contrôler l'ordre de révision dans une approche orientée contraintes. Nous avons montré expérimentalement que même si le comportement pathologique n'est

	AC-V	AC-V/B$^{\text{ary}}$ h		AC-C	AC-C/Binomial heap				AC-C/8 FIFO	
	FIFO	*dom*	$\frac{dom}{wdeg}$	FIFO	Πdom	$\Pi dom/w$	*eval*	$eval/w$	Arity	log *eval*
bigleq-400	651,0	666,8	667,4	643,3	58,5	231,3	**18,4**	222,3	**18,6**	**18,9**
bqwh-18-141-0-ext	41,8	37,4	43,8	36,0	31,7	52,4	33,4	53,6	35,3	**25,6**
bqwh-18-141-47-glb	94,8	90,3	93,2	**82,7**	**80,0**	**87,7**	**85,4**	89,7	**87,0**	**82,6**
crossword-lex-vg5-7	140,4	124,8	148,6	**87,5**	93,6	90,7	**81,3**	93,2	91,9	**80,9**
frb40-19-2-ext	57,5	45,0	60,6	41,7	**39,0**	85,2	**37,3**	87,8	41,6	**36,9**
js-e0ddr1	311,1	280,3	314,0	**189,0**	228,4	**195,7**	358,7	240,3	275,5	**187,0**
langford-3-13	109,9	104,4	117,2	45,5	**44,4**	45,0	**41,9**	74,3	**41,5**	**40,6**
lemma-24-3	68,9	70,1	69,7	**60,2**	**61,1**	**60,9**	**60,4**	**59,9**	**59,5**	**60,1**
os-taillard-5-100-4	504,6	473,3	509,1	**302,3**	337,0	**312,0**	449,5	408,9	357,0	**310,3**
queens-40	77,0	84,7	97,0	45,0	44,8	43,5	**38,9**	58,7	**35,6**	**36,3**
ruler-44-9-a3	21,0	17,6	21,1	8,0	8,1	8,4	7,3	12,0	**6,1**	**6,7**
scen11	811,0	653,3	711,8	767,1	**597,1**	**578,2**	**587,8**	**594,6**	779,6	**588,1**
series-20	15,3	14,5	16,4	**9,9**	10,4	10,8	**9,1**	11,2	**9,6**	**9,0**
tsp-20-193-ext	126,6	112,8	131,3	**82,0**	**83,6**	**86,1**	89,3	103,6	**79,5**	**84,5**

TABLE 4 – Temps de résolution (en secondes) avec l'heuristique d'instanciation $dom/ddeg$. Les meilleurs temps (dans une marge de 10 %) sont surlignés.

pas rencontré, une propagation orientée contraintes est meilleure dans une grande majorité des cas qu'une propagation orientée variables. La méthode que nous avons proposée est compétitive avec les approches retenues par les meilleurs solveurs disponibles, tout en étant plus évolutive : il est très probable qu'une heuristique capable de prendre en compte la probabilité qu'a une contrainte de supprimer des valeurs sera des plus intéressantes. Notre méthode permet de proposer de nouveaux évaluateurs basés sur cet aspect de la propagation. Ils restent encore à concevoir et à évaluer.

Références

[1] T. Balafoutis and K. Stergiou. Evaluating and improving modern variable and revision ordering strategies in CSPs. *Fundam. Inform.*, 102(3–4) :229–261, 2010.

[2] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global constraint catalog. Technical Report T2005-08, SICS, 2005.

[3] N. Beldiceanu and E. Contejean. Introducing Global Constraints in CHIP. *Mathl. Comput. Modelling*, 20(12) :97–123, 1994.

[4] C. Bessière, G. Katsirelos, N. Narodytska, and T. Walsh. Circuit complexity and decompositions of global constraints. In *Proc. IJCAI'09*, pages 412–418, 2009.

[5] C. Bessière and J.-C. Régin. AC for General Constraint Networks : Preliminary Results. In *Proc. IJCAI'97*, pages 398–404, 1997.

[6] C. Bessière, J.-C. Régin, R.H.C. Yap, and Y. Zhang. An Optimal Coarse-Grained Arc Consistency Algorithm. *Artificial Intelligence*, 165(2) :165–185, 2005.

[7] F. Boussemart, F. Hemery, C. Lecoutre, and L. Saïs. Boosting systematic search by weighting constraints. In *Proc. of ECAI'04*, pages 146–150, 2004.

[8] F. Boussemart, F. Hemery, and C. Lecoutre. Revision ordering heuristics for the CSP. In *Proc. CPAI'04 workshop held with CP'04*, pages 29–43, 2004.

[9] M.L. Fredman and R.E. Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *Journal of the ACM*, 34(3) :596–615, 1987.

[10] I.P. Gent, C. Jefferson, and I. Miguel. Minion : A Fast, Scalable, Constraint Solver. In *Proceedings of ECAI'06*, pages 98–102, 2006.

[11] F. Laburthe et al. CHOCO : Implementing a CP kernel. In *Proc. of the TRICS'2000 workshop held with CP'2000*, pages 71–85, 2000.

[12] M. Z. Lagerkvist and C. Schulte. Advisors for incremental propagation. In *Proc. of CP'07*, pages 409–422, 2007.

[13] M.K. Lagerkvist and C. Schulte. Propagator groups. In *Proc. of CP'2009*, pages 524–538, 2000.

[14] C. Lecoutre. STR2 : Optimized simple tabular reduction for table constraints. *Constraints*, 16(4) :341–371, 2011.

[15] C. Lecoutre and F. Hemery. A Study of Residual Supports in Arc Consistency. In *Proceedings of IJCAI'2007*, pages 125–130, 2007.

[16] C. Lecoutre and J. Vion. Enforcing Arc Consistency using Bitwise Operations. *Constraint Programming Letters*, 2 :21–35, 2008.

[17] A. López-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek. A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *Proc. IJCAI'03*, pages 245–250, 2003.

[18] A.K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1) :99–118, 1977.

[19] J.J. McGregor. Relational Consistency Algorithms and their Application in Finding Subgraph and Graph Isomorphisms. *Information Sciences*, 19 :229–250, 1979.

[20] S. Merchez, C. Lecoutre, and F. Boussemart. Abscon : a prototype to solve CSPs with abstraction. In *Proceedings of CP'01*, pages 730–744, 2001.

[21] C. Schulte et al. Generic Constraint Development Environment (Gecode). `http://www.gecode.org/`, 2005-2010.

[22] C. Schulte and P.J. Stuckey. Efficient Constraint Propagation Engines. *ACM Transactions on Programming Languages and Systems*, 31(1) :1–43, 2008.

[23] M. van Dongen, C. Lecoutre, and O. Roussel. Third International CSP Solvers Competition. `http://www.cril.univ-artois.fr/CPAI08`, 2008.

[24] P. van Hentenryck, Y. Deville, and CM. Teng. A Generic AC Algorithm and its Specializations. *Artificial Intelligence*, 57 :291–321, 1992.

[25] J. Vion. Constraint Satisfaction Problem for Java. `http://cspfj.sourceforge.net/`, 2006–2012.

[26] J. Vuillemin. A Data Structure for Manipulating Priority Queues. *Communications of the ACM*, 21 :309–314, 1978.

[27] R.J. Wallace and E.C. Freuder. Ordering Heuristics for Arc Consistency Algorithms. In *Proceedings of NCCAI'92*, pages 163–169, 1992.

Casser les symétries de variables dans un problème "presque" injectif

Philippe Vismara[1,2] **Rémi Coletta**[1]

[1] LIRMM, UMR5506 Université Montpellier II - CNRS, Montpellier, France
[2] MISTEA, UMR729 Montpellier SupAgro - INRA, Montpellier, France
{coletta,vismara}@lirmm.fr

Résumé

Une des techniques pour éliminer les symétries de variables est l'ajout de contraintes lexicographiques. Dans le cas général, le nombre de contraintes à ajouter au modèle pour éliminer toutes les symétries de variables est potentiellement exponentiel en n le nombre de variables [3]. Dans le cas particulier des problèmes injectifs (avec un AllDiff), le nombre de contraintes à ajouter est linéaire en le nombre de variables [8].

En se basant sur la contrainte globale de cardinalité [9], vue comme une généralisation de la contrainte AllDiff, nous caractérisons les problèmes "presque" injectifs par un paramètre μ correspondant au nombre de doublons.

Nous montrons ensuite que, pour ces problèmes "presque" injectifs, le nombre de contraintes à ajouter est majoré par $\binom{n}{\mu}$ et donc XP en termes de complexité paramétrée. Dans le cas où seules ν variables peuvent être affectées à un doublon, le nombre de contraintes est FPT en μ et ν.

Abstract

Lexicographic constraints are commonly used to break variable symmetries. In the general case, the number of constraint to be posted is potentially exponential in the number of variables. For injective problems (AllDiff), Puget's method[8] breaks all variable symmetries with a linear number of constraints.

In this paper we assess the number of constraints for "almost" injective problems. We propose to characterize them by a parameter μ based on Global Cardinality Constraint as a generalization of the AllDiff constraint. We show that for almost injective problems, variable symmetries can be broken with no more than $\binom{n}{\mu}$ constraints which is XP in the framework of parameterized complexity. When only ν variables can take duplicated values, the number of constraints is FPT in μ and ν.

1 Introduction

L'importance de la prise en compte des symétries dans la résolution d'un problème de satisfaction de contraintes est aujourd'hui largement reconnue. De nombreuses méthodes génériques on été développées pour éliminer ces symétries notamment de variables. Mais elles nécessitent généralement l'ajout d'un nombre de contraintes proportionnel au nombre de symétries qui peut s'avérer exponentiel. C'est le cas des contraintes lexicographiques [3] ou pour l'ajout dynamique de contraintes comme dans SBDS [5].

Mais sous certaines hypothèses, on peut se ramener à un nombre réduit de contraintes. Dans le cas des problèmes injectifs Pujet [8] a montré qu'il est possible d'éliminer les symétries de variables en posant un nombre linéaire de contraintes.

Entre ces deux extrêmes, cet article cherche à utiliser une approche de type complexité paramétrée pour évaluer le nombre de contraintes nécessaires à l'élimination des symétries de variables.

La notion de complexité paramétrée [4] permet de mesurer la complexité d'un problème NP-complet à l'aide d'une valeur k indépendante de la taille n des données. La complexité du problème est alors XP si elle s'exprime sous la forme $\mathcal{O}(n^k)$, voire FPT si elle est en $\mathcal{O}(f(k)n^c)$ où c est une constante et f une fonction quelconque, éventuellement exponentielle.

De nombreux problèmes de l'Intelligence Artificielle et plus particulièrement des CSP ont été analysés en termes de complexité paramétrée [6]. Par exemple, l'élimination des symétries de valeurs peut être réalisée avec une complexité paramétrée en $\mathcal{O}(2^k nd)$, où k est le nombre, potentiellement exponentiel, de symétries de valeurs [1].

De manière analogue, on peut bien sûr dire que le

nombre de contraintes nécessaires pour éliminer les symétries de variables est généralement FPT en le nombre de symétries.

L'objectif de cet article est d'essayer d'affiner ce critère, notamment dans le cas de problèmes "presque injectifs" .

2 Symétries dans les problèmes injectifs

2.1 symétries de variables

Une symétrie sur l'ensemble des variables $\{x_i\}_{i\in 1..n}$ est une permutation σ de $1..n$ telle que toute affectation $(x_i = v_i)_{i\in 1..n}$ est une solution si et seulement si $(x_{\sigma(i)} = v_i)_{i\in 1..n}$ en est une.

Une des démarches les plus courantes pour éliminer les symétries de variables consiste à poser des contraintes lexicographiques [3]. Étant donné un ordre $x_{i_1}, \dots x_{i_n}$ sur les variables, on pose, pour chaque symétrie de variable σ, une contrainte de la forme :

$$x_{i_1}, \dots, x_{i_n} \leq_{lex} x_{\sigma(i_1)}, \dots, x_{\sigma(i_n)} \qquad (1)$$

Malheureusement, le nombre de symétries (ou permutations) peut être exponentiel.

Généralement, un groupe $\mathcal{G}$ de symétries (ou permutations) est donné sous la forme d'un ensemble de générateurs S. On peut par exemple déterminer la taille et les générateurs du groupe de symétries (automorphismes) d'un graphe avec un logiciel comme Nauty [7] (ou encore Saucy). Ces logiciels sont très efficaces en pratique, comme l'écrit Brendan McKay : "*the theoretical worst-case efficiency of nauty is exponential. However, the worst cases are rather hard to find and for most classes of graphs nauty behaves as if it is polynomial.*"

Dans le cas de problèmes injectifs, on peut facilement montrer [8] que l'équation (1) se simplifie en $x_{i_k} \leq x_{\sigma(i_k)}$ où i_k est le plus petit indice tel que $\sigma(i_k) \neq i_k$. On aura donc moins de n^2 contraintes de différence pour l'ensemble des symétries.

Pour un indice i_k donné, les contraintes de la forme $x_{i_k} \leq x_{\sigma(i_k)}$ correspondent aux symétries σ qui sont des stabilisateurs de $i_1, .., i_{k-1}$ c.-à-d. au sous-groupe $\mathcal{G}^{[i_k]} = \{\sigma \in \mathcal{G} \mid \forall z < k,\ \sigma(i_z) = i_z\}$.

Plus précisément, on s'intéresse à toutes les images possibles de i_k par une symétrie qui laisse $i_1, .., i_{k-1}$ invariants. Cet ensemble d'images (ou *orbite*) peut se définir par $\Delta^{[i_k]} = \{\sigma(i_k) | \sigma \in \mathcal{G}^{[i_k]}\}$

Cette approche est intéressante parce qu'il est possible de déterminer tous les $\Delta^{[i_k]}$ sans énumérer $\mathcal{G}$, grâce à l'algorithme de Schreier-Sims.

2.2 L'algorithme de Schreier-Sims

On peut déjà simplifier le problème dans la mesure où il n'est pas nécessaire de considérer les n indices i_k mais seulement une partie de l'ensemble $1..n$.

En 1970, Sims a introduit la notion de *base* d'un groupe de permutations sur $1..n$. Une séquence $B = (\beta_1, \beta_2, ..., \beta_m)$, où $m \leq n$, est une base de $\mathcal{G}$ si aucune permutation de $\mathcal{G}$ n'est invariante pour B, à l'exception de l'identité. On parle de base parce que tout élément σ de $\mathcal{G}$ est parfaitement identifié par l'image $\sigma(B) = (\sigma(\beta_1), \sigma(\beta_2), ..., \sigma(\beta_m))$.

Une base B induit une chaîne de stabiliseurs $\mathcal{G} = \mathcal{G}^{[\beta_1]} \geq \mathcal{G}^{[\beta_2]} \geq \dots \geq \mathcal{G}^{[\beta_m]} \geq \{id.\}$ puisque chaque $\mathcal{G}^{[\beta_{k+1}]}$ (les stabilisateurs de $\beta_1, .., \beta_k$) est un sous-groupe de $\mathcal{G}^{[\beta_k]}$.

Si en plus ce sont des sous-groupes propres (non égaux), on dit que B est *non-redondante* et on montre que $2^{|B|} \leq |\mathcal{G}|$ c'est-à-dire $|B| \leq \log_2(|\mathcal{G}|)$

Étant donné un ensemble générateur S de $\mathcal{G}$, l'algorithme de Schreier-Sims construit incrémentalement une base B non-redondante et ajoute des générateurs à S afin que $S \cap \mathcal{G}^{[\beta_k]}$ soit un générateur de $\mathcal{G}^{[\beta_k]}$ (on parle alors de *strong generating set*).

Il détermine également l'orbite $\Delta^{[\beta_k]}$ correspondant à chaque β_k. Enfin, l'algorithme choisit un représentant (on parle de *transversal*) pour chaque valeur γ de $\Delta^{[\beta_k]}$, c'est-à-dire une permutation de $\mathcal{G}^{[\beta_k]}$, notée $u^{\gamma}_{\beta_k}$, qui associe β_k à γ (tout en laissant les $\beta_1, ..., \beta_{k-1}$ invariants).

Il est alors possible d'énumérer efficacement tous les éléments de $\mathcal{G}$, en utilisant une combinaison de représentants de chaque β_k. Si $U^{[k]} = \{u^{\gamma}_{\beta_k} | \gamma \in \Delta^{[\beta_k]}\}$ désigne l'ensemble de ces représentants pour β_k, on a $\mathcal{G} = \{\upsilon_{\beta_1} \circ \upsilon_{\beta_2} \circ \dots \circ \upsilon_{\beta_m} \mid \forall k \in 1..m, \upsilon_{\beta_k} \in U^{[k]}\}$

Il existe plusieurs variantes de l'algorithme de Schreier-Sims et une vaste littérature sur le sujet [2, 10]. La version déterministe la plus simple a une complexité en temps de $\mathcal{O}(n^2 \log^3 |\mathcal{G}| + |S| n^2 \log |\mathcal{G}|)$ et $\mathcal{O}(n^2 \log |\mathcal{G}| + |S| n)$ en mémoire.

En prenant pour base $B = (1, 2, ..., n)$ la variante proposé par Jerrum a une complexité $\mathcal{O}(n^5)$ en temps et $\mathcal{O}(n^2)$ en espace.

2.3 Un nombre polynomial de contraintes

Grâce à l'algorithme de Schreier-Sims on peut construire, en temps polynomial, les orbites $\Delta^{[\beta_k]}$ à partir d'un ensemble générateur S donné [1].

Chaque symétrie σ dans $\mathcal{G}$ appartient nécessairement à un sous-groupe $\mathcal{G}^{[\beta_k]}$. On peut donc remplacer la contrainte (1) par l'inégalité $x_{\beta_k} < x_{\sigma(\beta_k)}$ sachant

1. ou calculé avec *Nauty* en temps éventuellement exponentiel

que $\sigma(\beta_k) \in \Delta^{[\beta_k]}$. On en déduit un nombre polynomial d'inégalités :

$$\forall \beta_i \in B,\ \forall \gamma \in \Delta^{[\beta_i]}, \gamma \neq \beta_i,\ \text{on pose}\ \ x_{\beta_i} < x_\gamma \quad (2)$$

Le nombre d'inégalités est égal à $\sum_{\beta_i \in B}(|\Delta^{[\beta_i]}| - 1)$ donc en $\mathcal{O}(n\, log_2|\mathcal{G}|)$ ou encore $\mathcal{O}(n^2)$

Dans [8], il est montré qu'on peut se limiter à une inégalité pour chaque γ, en ne considérant que $r(\gamma)$, le plus grand des β_i (différent de γ) tel que $\gamma \in \Delta^{[\beta_i]}$.

Formellement, soit $\mathcal{R}_\gamma = \{\beta_i \in B \setminus \{\gamma\} \mid \gamma \in \Delta^{[\beta_i]}\}$. Si $\mathcal{R}_\gamma \neq \emptyset$ on pose $r(\gamma) = \max(\mathcal{R}_\gamma)$ sinon $r(\gamma) = \gamma$.

On peut montrer [2] que les équations (2) sont équivalentes à un nombre linéaire de contraintes :

$$\forall \gamma \in 1..n \text{ tel que } r(\gamma) \neq \gamma,\ \text{on pose}\ \ x_{r(\gamma)} < x_\gamma \quad (3)$$

3 Gcc : relaxation du Alldiff aux problèmes "presque" injectifs

La `Gcc` (Global Cardinality Constraint) a été introduite dans [9]. C'est une contrainte largement utilisée dans la modélisation de problèmes industriels et elle est disponible dans la plupart des résolveurs de contraintes existants. Elle exprime une propriété très générique : elle contraint le nombre d'occurrences de chaque valeur présente dans les domaines. En d'autres termes, cette contrainte traite globalement une conjonction de `AtLeast` (imposant qu'une valeur v apparaisse au moins un certain nombre de fois) et `AtMost` (imposant qu'une valeur v apparaisse au plus un certain nombre de fois).

Définition 1 *Une contrainte* `Gcc`*, notée* $Gcc(X, lb, up)$ *porte sur un ensemble de variables* X *et de 2 fonctions* lb *et* ub *de* $\bigcup_{x\in X} D(x)$ *dans* $\mathcal{N}$*. La contrainte* `Gcc` *est vérifiée si pour chaque valeur* $v \in \bigcup_{x\in X} D(x)$ *le nombre de variables de* X *assignées à* v *est compris entre* $lb(v)$ *et* $ub(v)$*.*

Soit $N =< X, D, C >$ un réseau de contraintes avec $Gcc(X, lb, ub) \in C$. On pose $\mu = 1 + \sum_{v\in D}(ub(v) - 1)$

Si $\mu = 1$, alors la *Gcc* est un Alldiff, on a un problème injectif. Si μ est "petit", alors on a un problème "presque" injectif. Le paramètre μ permet donc de mesurer l'écart, en nombre de doublons autorisés, par rapport à un problème parfaitement injectif.

4 Généralisation aux problèmes "presque" injectifs

Considérons un problème presque injectif admettant au plus μ doublons.

Pour chaque symétrie de variables $\sigma \in \mathcal{G}^{[\beta_k]}$, la contrainte lexicographique (1) n'est plus équivalente à $x_{\beta_k} < x_{\sigma(\beta_k)}$ puisque $x_{\beta_k} = x_{\sigma(\beta_k)}$ devient possible. En cas d'égalité, tester la contrainte lexicographique revient à déterminer le prochain $\beta_z > \beta_k$ tel que $x_{\beta_z} \neq x_{\sigma(\beta_z)}$, avec nécessairement $\sigma(\beta_z) \neq \beta_z$.

On peut supposer ici que $|B| = n$. Sinon il suffit d'ajouter à la fin de la base les valeurs manquantes dans un ordre quelconque. Les générateurs calculés par l'algorithme de Schreier-Sims restent valables pour cette base étendue.

Formellement, étant donnée une symétrie de variables $\sigma \in \mathcal{G}^{[\beta_k]}$, considérons la suite croissante $i^1_\sigma, i^2_\sigma, \dots i^t_\sigma$ des indices de B pour lesquels $\sigma(i) \neq i$. Puisque $\sigma \in \mathcal{G}^{[\beta_k]}$ on a $i^1_\sigma = \beta_k$.

Soit ρ le plus petit indice de $1..t$, s'il existe, tel que :

$$|\{i^z_\sigma\}_{z\in 1..\rho} \cup \{\sigma(i^z_\sigma)\}_{z\in 1..\rho}| > \mu$$

Si t est trop petit pour permettre de couvrir les μ variables, on pose $\rho = t \leq \mu$.

Sinon, dans le cas général : $\frac{\mu}{2} < \rho \leq \mu + 1$.

Pour un problème injectif, la contrainte lexico (1) était remplacée par la contrainte $x_{i^1_\sigma} < x_{\sigma(i^1_\sigma)}$

Dans le cas où μ valeurs identiques sont possibles, la contrainte lexico (1) devient :

$$x_{i^1_\sigma}, x_{i^2_\sigma}, \dots, x_{i^\rho_\sigma} \leq_{lex} x_{\sigma(i^1_\sigma)}, x_{\sigma(i^2_\sigma)}, \dots, x_{\sigma(i^\rho_\sigma)} \quad (4)$$

Au final, on aura donc au plus $\binom{n}{\mu+1}$ contraintes (4) pour l'ensemble des symétries.

On obtient ainsi un nombre de contraintes lexicographiques qui n'est plus potentiellement exponentiel mais XP en fonction de μ puisque $\binom{n}{\mu} < n^\mu$

Pour construire l'ensemble des contraintes (4), il faut énumérer l'ensemble $\mathcal{G}$ des symétries ce qui peut être réalisé en $\mathcal{O}(n|\mathcal{G}|)$, grâce aux ensembles $U^{[k]}$ calculés par l'algorithme de Schreier-Sims.

On peut remarquer que chaque contrainte (4) se décompose en contraintes de la forme :

$$x_{i^1_\sigma} \leq x_{\sigma(i^1_\sigma)} \quad (5a)$$

$$x_{i^1_\sigma} = x_{\sigma(i^1_\sigma)} \rightarrow x_{i^2_\sigma} \leq x_{\sigma(i^2_\sigma)} \quad (5b)$$

$$\dots$$

$$x_{i^1_\sigma} = x_{\sigma(i^1_\sigma)} \wedge \dots \wedge x_{i^{\rho-1}_\sigma} = x_{\sigma(i^{\rho-1}_\sigma)} \rightarrow x_{i^\rho_\sigma} \leq x_{\sigma(i^\rho_\sigma)} \quad (5c)$$

Il y aura $\sum_{\beta_i \in B}(|\Delta^{[\beta_i]}| - 1)$ contraintes de type (5a), au plus $\binom{n}{4}$ contraintes de type (5b) et finalement au plus $\binom{n}{\mu+2}$ contraintes de type (5c).

C'est une majoration très grossière et dans le pire des cas. En pratique on peut éliminer des contraintes. Par exemple, si on a posé une contrainte (5a) de

2. Chaque $x_{\beta_i} < x_\gamma$ est équivalent à la série d'inégalités $x_{\beta_i = r(\dots r(r(\gamma)))} < \dots < x_{r(r(\gamma))} < x_{r(\gamma)} < x_\gamma$

la forme $x_a \leq x_b$ il est inutile de poster toutes les contraintes finissant par « $\rightarrow x_a \leq x_b$ ».

Par ailleurs, les contraintes (5a) peuvent être remplacées par un nombre linéaire de contraintes de la forme $x_{r(j)} \leq x_j$.

5 Discussion

Nous avons établi que dans le cas d'un problème presque injectif, le nombre de contraintes lexicographiques nécessaires pour éliminer toutes les symétries de variables est en $\mathcal{O}(\binom{n}{\mu})$.

Ce nombre dépend des symétries mais aussi de la base B utilisée. Supposons que δ soit le plus petit indice (dans l'ordre de B) d'une variable x_δ dont la valeur n'est pas unique. Pour chaque symétrie $\sigma \in \mathcal{G}^{[\beta_k]}$ telle que $i^1_\sigma = \beta_k < \delta$, tout se passe comme si le problème était injectif. Dans le cas extrême où $\delta > |B|$ (avec $|B| < n$), on se ramène au cas injectif.

On pourrait donc envisager d'adapter la base B pour que les indices des variables ayant une valeur répétée soient placés à la fin de B.

Une adaptation dynamique de la base serait probablement très coûteuse. D'une part parce que les contraintes de type (2) ne pourraient être activées dynamiquement que si on est certain d'avoir $\beta_k < \delta$. Par ailleurs, il est possible de réordonner B pour déplacer les indices correspondant à des doublons vers la fin. Mais la simple permutation de deux indices contigus coûte $\mathcal{O}(n^4)$[2].

De façon statique, on peut imaginer des cas particuliers où les doublons ne peuvent concerner qu'un sous ensemble de variables. Si on peut construire une base B avec les indices des autres variables, on peut éliminer les symétries comme dans le cas injectif.

Supposons enfin qu'il n'y ait que ν variables pouvant être affectées à un doublon. Dans ce cas, la base B peut être construite de telle sorte que les indices de ces ν variables soient placés à la fin. On est alors ramené au cas injectif pour toute symétrie σ telle que $i^1_\sigma < \beta_{n-\nu}$ et l'ensemble des symétries de $\mathcal{G} \setminus \mathcal{G}^{[\beta_{n-\nu}]}$ se ramène à un nombre linéaire de contraintes. Seules les symétries de $\mathcal{G}^{[\beta_{n-\nu}]}$ nécessiteront une contrainte lexico (4). Puisque ces contraintes ne concernent que ν variables, leur nombre est majoré par $\binom{\nu}{\mu}$. Le nombre total de contrainte est donc dans ce cas en $\mathcal{O}(\binom{\nu}{\mu} + n)$ c'est-à-dire FPT en ν et μ.

Influence de domaines hétérogènes

Dans la section 4, nous avons donné une borne théorique du nombre de contraintes à poser. La majoration ne tient pas compte des domaines initiaux des variables, dans le cas où ils ne sont pas tous égaux.

En effet, si $D(x_{i^1_\sigma}) \cap D(x_{\sigma(i^1_\sigma)}) = \emptyset$, il est inutile de poser la contrainte 5b et toutes les contraintes qui contiennent $x_{i^1_\sigma} = x_{\sigma(i^1_\sigma)}$ en partie gauche.

La combinaison de la contrainte de cardinalité et de la répartition initiale des domaines peut aussi interdire des combinaisons d'égalité : Par exemple si $D(x_1) \cap D(x_2) = \{v\} = D(x_3) \cap D(x_4)$ et $ub(v) = 3$, alors la `Gcc` interdit de fait d'avoir $x_1 = x_2$ et $x_3 = x_4$ simultanément. Donc toutes les contraintes 5c avec $x_1 = x_2 \wedge x_3 = x_4$ en partie gauche n'ont pas besoin d'être posées.

Plus généralement, lors de la génération des contraintes d'un réseau $N =< X, D, C >$. Avant de poser la contrainte $x_{i^1_\sigma} = x_{\sigma(i^1_\sigma)} \wedge \ldots \wedge x_{i^{\rho-1}_\sigma} = x_{\sigma(i^{\rho-1}_\sigma)} \rightarrow x_{i^\rho_\sigma} \leq x_{\sigma(i^\rho_\sigma)}$, nous proposons de tenter de résoudre le problème $N' =< X, D, C' >$ avec $C' = \{x_{i^1_\sigma} = x_{\sigma(i^1_\sigma)}, \ldots, x_{i^{\rho-1}_\sigma} = x_{\sigma(i^{\rho-1}_\sigma)}\} \cup \{Gcc\}$, le sous problème constitué la `Gcc` et des égalités de la gauche de cette contrainte. Si N' est insatisfiable, alors on ne pose pas la contrainte.

Ce test est polynomial, car la classe des réseaux de contraintes composés d'une contrainte `Gcc` et de contraintes d'égalités est polynomiale pour le problème de satisfaction. L'idée du codage consiste en une modification du problème de flot utilisé pour résoudre la contrainte de `Gcc` [9]. Pour toute paire de variables (x_i, x_j) telle que $x_i = x_j$, on supprime les nœuds x_i et x_j, ainsi que leur arêtes et on ajoute un noeud $x_{(i,j)}$ et $\forall v \in D(x_i) \cap D(x_j)$, on ajoute une arrête $(x_{(i,j)}, v)$ avec $(0, 2)$ comme flots minimum et maximum.

Cette démarche fournit ainsi un moyen supplémentaire de réduire le nombre de contraintes pour les problèmes presque injectifs, en complément des résultats que nous avons établis en complexité paramétrée.

6 Conclusion

Nous avons montré qu'il est possible d'éliminer les symétries de n variables d'un problème presque injectif en posant moins de $\binom{n}{\mu}$ contraintes lexicographiques, où μ correspond au nombre de doublons. On obtient bien un nombre XP de contraintes lexicographiques.

Dans le cas où il n'y a que ν variables qui peuvent être affectées à un doublon, le nombre des contraintes posées est en $\mathcal{O}(\binom{\nu}{\mu} + n)$ donc FPT.

Lorsque les domaines des variables sont hétérogènes, nous avons montré qu'un test polynomial (basé sur la `Gcc`) permet d'écarter des contraintes lexicographiques inutiles.

Références

[1] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, C.-G. Quimper, and T. Walsh. The parameterized

complexity of global constraints. In *Proceedings AAAI*, pages 235–240, 2008.

[2] Gregory Butler. *Fundamental Algorithms for Permutation Groups*, volume 559 of *Lecture Notes in Computer Science*. Springer, 1991.

[3] James Crawford, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *KR*, pages 148–159. Morgan Kaufmann, 1996.

[4] Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. Parameterized complexity : A framework for systematically confronting computational intractability. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 49, pages 49–99, 1997.

[5] Ian P. Gent, Warwick Harvey, and Tom Kelsey. Groups and constraints : Symmetry breaking during search. In *CP 2002*, volume 2470 of *Lecture Notes in Computer Science*, pages 415–430, 2002.

[6] Georg Gottlob and Stefan Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems. *Comput. J.*, 51(3) :303–325, 2008.

[7] B. D. McKay. *nauty user's guide (version 2.4)*. Technical report, Australian National University, Computer Science Department, `http://cs.anu.edu.au/~bdm/nauty/`, 2009.

[8] Jean-François Puget. Breaking symmetries in all different problems. In *Proceedings IJCAI'05*, pages 272–277, 2005.

[9] J-C. Régin. Generalized arc consistency for global cardinality constraint. *AAAI*, pages 209–215, 1996.

[10] A. Seress and Á. Seress. *Permutation group algorithms*. Cambridge tracts in mathematics. Cambridge University Press, 2003.

Index des auteurs

Avec le soutien de