Autour du problème de satisfaction de contraintes

Amit Bellicha, Bertrand Neveu, Brigitte Trousse,
Christian Bessière, Christine Gaspin, David Lesaint,
Fabrice Bouquet, Florence Dupin de Saint-Cyr, Gilles Trombettoni,
Gérard Verfaillie, Hélène Fargier, Jean-Charles Régin,
Jean-Pierre Rellier, Jérôme Amilhastre, Jérôme Lang,
Khaled Ghedira, Marie-Catherine Vilarem, Martin Cooper,
Olivier Lhomme, Philippe Charman, Philippe David,
Philippe Janssen, Philippe Jégou, Pierre Berlandier,
Roger Martin-Clouaire, Thomas Schiex, Tibor Kökény

10 avril 1997

1 Introduction

Le cadre des problèmes de satisfaction de contraintes ou CSP (pour *Constraint Satisfaction Problems*) a pour finalité l'expression et la résolution de problèmes faisant intervenir des contraintes. De façon très générale, une contrainte correspond à l'énoncé d'une propriété relative à certaines caractéristiques de différents objets : propriété physique (spatio-temporelle. . .) nécessairement satisfaite par les objets considérés ou propriété désirée par l'utilisateur par exemple.

Du fait de sa généralité, il est assez naturel que la notion de contrainte soit fréquemment utilisée et ait reçu beaucoup d'attention de la part des communautés de l'intelligence artificielle ou de la recherche opérationnelle, toutes deux préoccupées par la *représentation* et la *résolution* de problèmes. Face aux illustres prédécesseurs que sont la programmation linéaire (en nombres entiers) ou la logique propositionnelle, le cadre CSP offre un ensemble de caractéristiques pragmatiques intéressantes :

– le langage d'expression des contraintes est général et souple. De ce fait, le problème de la linéarité des contraintes ne se pose pas et l'expression est généralement plus concise qu'en utilisant le langage traditionnel de la logique propositionnelle ;

– de nombreux algorithmes existent pour résoudre une grande variété de problèmes, de complexités variées.

Naturellement, tout comme pour ces prédécesseurs, certains algorithmes font face à la généralité du cadre d'expression : de nombreux problèmes sont NP-difficiles.

Plus précisément, un CSP est défini par :

– un ensemble $X = \{x_1, \ldots, x_n\}$ de n variables. Chaque variable représente une caractéristique des objets apparaissant dans le problème réel (position spatiale, date de démarrage d'une tâche. . .) ;

– un ensemble $D = \{d_1, \dots, d_n\}$ de n domaines. Le domaine d_i est associé à la variable x_i et représente l'ensemble (habituellement fini) des valeurs ou états que peut prendre la variable. On note d le cardinal du plus grand domaine ;

– un ensemble $C = \{c_1, \dots, c_e\}$ de e contraintes. Une contrainte c_k est définie par l'ensemble des variables sur lesquelles elle pèse et par l'ensemble des combinaisons de valeurs qu'elle autorise (relation R_k définie par un sous-ensemble du produit cartésien des domaines des variables contraintes. Il peut s'agir d'une définition en intension ou en extension). L'arité d'une contrainte est égale au nombre de variables sur lesquelles elle pèse.

On note habituellement r le maximum des arités des contraintes apparaissant dans un CSP. Les CSP binaires (ceux pour lesquels toutes les contraintes ont une arité égale à 2), correspondent ainsi à $r = 2$. Dans ce cas, on notera c_{ij} la contrainte pesant sur la paire de variables $\{x_i, x_j\}$ et R_{ij} sa relation associée.

Le problème le plus souvent considéré dans ce cadre est le problème de satisfaction, consistant à chercher une valeur pour chacune des variables de façon à ce que toutes les contraintes soient simultanément satisfaites. Ce problème est NP-difficile (par restriction au problème de k-coloriage d'un graphe).

Malgré la grande généralité offerte par le cadre CSP quant à l'expression des contraintes, l'application à des problèmes réels montre différentes limitations auxquelles le groupe a tenté d'offrir des réponses :

– traitement de problèmes dynamiques dans lesquels la définition du problème est modifiée incrémentalement et pour lesquels il faudrait définir des algorithmes incrémentaux dans leur fonctionnement ;

– recherche et manipulation d'ensembles de solutions plutôt qu'une seule solution ;

– traitement du problème de l'absence de solution, souvent lié à l'expression sous forme de contraintes, de propriétés qu'il n'est pas absolument nécessaire de satisfaire (préférences) ou dont l'existence est incertaine ;

– expression et traitement de problèmes « hiérarchiques » pour lesquels, par exemple, l'existence de variables ou de contraintes est liée à la valeur d'une autre variable ;

– enfin, prise en compte des caractéristiques fines des CSP traités pour améliorer l'efficacité des algorithmes, problème qui reste central dans la communauté.

Les sections qui suivent présentent les approches techniques considérées par les membres du projet pour répondre à ces limitations.

2 Problèmes dynamiques

2.1 Le problème CSP dynamique

Origine – S'il est vrai que le formalisme et les algorithmes CSP permettent de représenter et de traiter de nombreux problèmes d'intelligence artificielle et de recherche opérationnelle (planification, ordonnancement, conception, diagnostic...), il reste un aspect qui, jusqu'à ces dernières années, n'avait reçu qu'une attention limitée, à savoir la nature *dynamique* de la plupart de ces problèmes, qui peut provenir :

- de l'*environnement* : exemple des problèmes de planification ou d'ordonnancement, où l'ensemble des actions à planifier et l'état du système physique permettant de les réaliser peuvent évoluer et où l'exécution est rarement conforme à la prévision (effets des actions, durée, consommation en ressources, pannes...) ;
- de l'*utilisateur* : exemple des problèmes de conception interactive, où ce dernier travaille par explorations successives de spécifications différentes ;
- plus généralement, d'*autres agents* dans un système distribué : exemple des problèmes de conception distribuée, où les décisions d'un agent modifient les spécifications des problèmes traités par d'autres.

La notion de CSP dynamique a été introduite par [22] pour représenter de telles situations : un CSP dynamique est simplement une *séquence* de CSP où chaque élément de la séquence diffère du précédent par l'*ajout* et/ou le *retrait* de certaines contraintes. On peut en effet aisément montrer qu'à condition d'inclure les domaines (contraintes unaires) dans l'ensemble des contraintes, cette définition couvre l'ensemble des modifications possibles d'un CSP.

Besoins – Dans ce cadre, le premier besoin est un besoin d'*efficacité*, qui n'est pas nouveau dans le domaine des CSP, mais qui peut être encore plus marqué dans un contexte dynamique : exemple des problèmes de planification ou d'ordonnancement en boucle fermée sur l'environnement, où les exigences de l'action imposent, en cas de perturbation, d'être capable de délivrer rapidement une nouvelle solution.

Le second besoin est un besoin de *stabilité* des solutions successives, qui peut être important, au cas où un certain travail (organisation, commandes...) a été engagé sur la base de la solution précédente.

Problème technique – Pour répondre à ces besoins, la démarche générale consiste à *mémoriser* des résultats (solution, ensemble de solutions, filtrage par cohérence locale, contraintes déduites...) et à les *réutiliser* dans le cadre de problèmes proches.

On peut tout de suite noter que tout retrait de contrainte conserve les solutions et que tout ajout conserve les déductions, mais que l'inverse n'est pas vrai : un ajout de contrainte peut invalider une solution et un retrait de contrainte invalider une déduction.

Le problème est donc de développer des méthodes permettant de maintenir solutions et/ou déductions quelle que soit la nature du changement : ajout et/ou retrait de contrainte.

2.2 Maintien de cohérence

Les méthodes de filtrage sont largement utilisées pour simplifier un CSP et réduire a priori l'espace de recherche. Elles s'appuient sur des notions de cohérence locale (k-cohérence) pour produire des contraintes qui sont conséquences des contraintes explicites du problème. La méthode la plus utilisée (celle qui produit généralement le meilleur résultat au moindre coût) est le filtrage par arc-cohérence, qui permet de retirer des domaines de chacune des variables certaines des valeurs qui ne participent à aucune solution (production de contraintes unaires). Nombreux sont les algorithmes qui ont été proposés pour filtrer un CSP par arc-cohérence (AC3, AC4, AC6...). Le problème est de les adapter au cadre dynamique, pour éviter de refaire l'ensemble du filtrage après chaque modification du CSP.

Comme cela a déjà été noté, seul le retrait de contrainte pose problème. Dans ce cas, la démarche consiste à limiter les réintroductions de valeur et la méthode globale comprend trois phases : *réintroduction de valeurs*, *propagation des réintroductions* et *nouveau filtrage*.

Les méthodes proposées peuvent être classées en deux groupes selon qu'elles utilisent ou non des *justifications* de retrait de valeur pour limiter les réintroductions futures.

Avec justifications – La première approche consiste à mémoriser, pour chaque retrait de valeur, la contrainte qui a justifié ce retrait et à utiliser cette information pour limiter les réintroductions de valeurs en cas de retrait de contrainte. Elle a été développée, implémentée et expérimentée sur la base de l'algorithme AC4 [9], puis récemment AC6 [20]. La comparaison avec l'exécution des algorithmes statiques sur chaque nouveau CSP montre des gains appréciables en termes d'efficacité.

Sans justifications – La seconde approche utilise la définition des contraintes pour limiter les réintroductions de valeur. Elle a été développée, implémentée et expérimentée sur la base d'AC3 [5]. Par rapport à l'approche précédente, elle a le défaut d'être moins efficace, mais le double avantage de ne requérir aucune mémorisation et de permettre la prise en compte des changements plus fins (modification de la définition d'une contrainte)

2.3 Maintien de solution

Comme pour le maintien de cohérence, les méthodes proposées peuvent être classées en deux groupes :

- la première approche consiste à améliorer les méthodes de recherche classiques à base de backtrack (recherche systématique) grâce à la production, la mémorisation et la réutilisation de *contraintes déduites justifiées*. La démarche est proche de celle du maintien de cohérence à base de justifications ;
- la seconde approche consiste à développer des méthodes dites de *réparation* (systématiques ou non), permettant de produire une solution pour un CSP à partir de n'importe quelle affectation de l'ensemble des variables et donc, plus particulièrement, à partir d'une solution produite pour un CSP précédent.

Par réutilisation de contraintes – La première approche, développée dans [60, 59] et désignée sous le terme de *Nogood Recording*, s'appuie sur une règle d'inférence simple, proche de la règle $H5$ utilisée par les *ATMS*, et sur laquelle s'appuient également les algorithmes de filtrage de k-consistance [19]. Il est ainsi possible de produire des *nogoods* au cours d'une recherche classique à base de backtrack. Les *nogoods* produits sont des affectations partielles de l'ensemble des variables qui ne peuvent en aucun cas être prolongées en une solution. Ils sont utilisés pour réduire l'espace de recherche courant, mais aussi, grâce aux justifications qui leur sont attachées, pour réduire l'espace de recherche de tout CSP voisin. La limitation de l'arité des *nogoods* mémorisés permet de borner polynomialement l'espace mémoire requis.

Sur des problèmes statiques, aussi bien que dynamiques, cette méthode a montré son efficacité (nombre de vérifications de contraintes, temps cpu), en particulier sur les problèmes intermédiaires et sur-contraints. Dans le cas statique, l'approche proposée est très proche du « *Jump-back learning* » proposé dans [37].

Par réutilisation de solution – La seconde approche regroupe une plus grande diversité de méthodes, certaines développant une recherche systématique de solution, d'autres une recherche non systématique.

Parmi les premières, on peut citer la méthode développée dans [68] et désignée sous le terme de *Local Changes*. S'appuyant sur un mécanisme de *backtrack* particulier, qui, suite à l'affectation d'une valeur à une variable x_i, désaffecte, puis réaffecte l'ensemble des variables dont l'affectation est incompatible avec celle de x_i. Elle permet de produire une solution à partir de n'importe quelle affectation (vide, partielle ou complète). Sur des problèmes statiques, aussi bien que dynamiques, elle a montré son efficacité, en particulier sur les problèmes sous et sur-contraints. Sur les problèmes dynamiques sous-contraints, elle a montré aussi de bons résultats en termes de stabilité (nombre de variables dont l'affectation est modifiée).

Entre les premières et les secondes, on peut citer les méthodes présentées dans [63, 62], à base de *propagation de valeurs*. Ces méthodes utilisent des mécanismes associés à chaque contrainte, qui permettent de réagir à la modification de la valeur d'une variable liée par la contrainte et de retrouver la cohérence en modifiant la valeur d'une autre variable liée par la même contrainte. Constatant que ces mécanismes ont le défaut d'être incomplets, diverses propositions ont été faites pour leur donner un caractère complet (recherche systématique).

Parmi les secondes, on peut citer la méthode développée dans [39, 41, 40], qui utilise une représentation *multi-agent* des CSP (un agent par variable et par contrainte) et un *recuit simulé* distribué sur chaque variable, pour produire une affectation maximisant le nombre de contraintes satisfaites. Développée dans le cadre des CSP statiques, elle a été étendue à celui des CSP dynamiques et a montré de bons résultats à la fois en termes d'efficacité et de stabilité, en particulier sur les problèmes sous-contraints.

À noter la proximité entre ces méthodes et les nombreuses méthodes de réparation de solution présentes dans la littérature récente [52, 38].

Par réutilisation de contraintes et de solution – On peut enfin citer l'extension de l'algorithme *Dynamic Backtracking* [42] au cadre des CSP dynamique [69]. Cette extension tire parti des deux approches précédentes dans la mesure où elle réutilise contraintes et solutions, obtenues lors des résolutions précédentes.

L'algorithme initial, développé dans le cadre des CSP statiques, utilise un mécanisme de *backtrack* particulier et des *explications* pour les *éliminations de valeur* (ensemble de variables précédemment affectées responsables de cette élimination). Dans le cadre des CSP dynamiques ce mécanisme de *backtrack* permet, d'une part, la réutilisation de solution ; l'introduction des contraintes du problème dans les explications des éliminations de valeurs (à côté des variables précédemment affectées) permet, d'autre part, la réutilisation de ces éliminations (dans une démarche identique à celle du *Nogood Recording*).

Les résultats obtenus sont ceux des méthodes à base de réutilisation de solution sur les problèmes sous-contraints (efficacité et stabilité) et ceux des méthodes à base de réutilisation de contraintes sur les problèmes intermédiaires et sur-contraints (efficacité).

2.4 Maintien de solution proche

L'expérience montre que les méthodes de réparation, réutilisant des solutions précédentes, fournissent généralement de bons résultats en termes de stabilité (solution proche de la solution précédente). Elles ne fournissent cependant pas la solution la plus proche. D'autres méthodes ont dû être développées pour répondre à l'objectif de minimisation du nombre de variables dont l'affectation est modifiée. Citons celle présentée dans [3] qui traite des CSP binaires acycliques, puis des CSP binaires de graphe quelconque, avec utilisation de la méthode dite *coupe-cycle* [21]. On peut noter que le critère de distance utilisé est agrégeable avec d'autres critères de type *coût* propres à l'application (CSP avec préférences additives).

2.5 Utilisation de BDD

Une autre approche du traitement de la dynamicité repose sur l'utilisation de BDD *Binary Decision Diagrams* [12], qui permettent de représenter avec concision l'arbre de décision associé à une fonction booléenne.

Un BDD est un graphe orienté sans circuit possédant une racine et deux puits. Les sommets internes sont étiquetés par les variables propositionnelles et les arcs par les affectations (0 ou 1) de leurs variables origines. Les puits sont étiquetés par les valeurs de vérité (0 et 1) de la fonction représentée. Tout chemin de la racine vers un puits correspond à une interprétation (éventuellement partielle) des variables ; la valeur de vérité de la fonction correspondant à cette interprétation est donnée par l'étiquetage du sommet d'arrivée. Dans le pire des cas, la taille d'un BDD représentant une fonction à n variables est égale à 2^n. En pratique, cette taille est très souvent bien inférieure.

L'utilisation des BDD dans le cadre CSP est possible de deux façons différentes : par extension du cadre BDD des variables binaires aux variables n-aires (voir [67, 2] pour des approches similaires mais s'appuyant sur le formalisme des automates d'états finis déterministes et non-déterministes respectivement) ou par passage du cadre CSP au cadre booléen [56]. C'est la seconde voie qui a été choisie dans [43].

La construction d'un BDD est incrémentale : utilisation des BBD représentant deux fonctions booléennes pour construire le BDD représentant la conjonction de ces deux fonctions. L'ajout de contrainte est donc sans problème. Pour le retrait de contrainte, la solution développée dans [43] consiste à traiter le retrait de façon préventive lors de l'ajout : soit une contrainte C et la fonction booléenne f associée; une variable booléenne c est créée pour représenter l'appartenance ou non de la contrainte au CSP courant et la fonction booléenne ajoutée au BDD n'est pas f, mais $(c \rightarrow f)$, ce qui permet ensuite un traitement extrêmement aisé de la dynamicité : $c = 1$ implique la satisfaction de la contrainte, $c = 0$ n'implique rien.

Il a été établi dans [11] que l'ajout de m variables booléennes représentant les contraintes ne faisait pas passer la taille du BDD dans le pire des cas de 2^n à 2^{n+m}, mais simplement à $m.2^n$, ce qui donne une complexité du même ordre que celle des méthodes de recherche arborescente, qui ne fournissent pas les mêmes informations et ne traitent pas aussi aisément la dynamicité. Les premières expériences, même si elles ont été réalisées sur des problèmes encore de taille modeste, montrent la viabilité de l'approche.

À noter aussi que les BDD peuvent aussi être vus comme un outil pour représenter l'ensemble des solutions de façon compacte (voir section 3) et pour traiter les préférences (voir section 4, il suffit de considérer une valuation des arcs étiquetés à 0 issus des sommets étiquetés par les contraintes, puis d'utiliser des algorithmes de recherche de chemin optimal).

3 Ensemble de solutions

Afin de pouvoir simplement trouver une (ou plusieurs) nouvelle(s) solution(s) suite à une modification du CSP, il est envisageable de maintenir l'ensemble des solutions du CSP. Une limitation technique essentielle de cette approche réside dans la taille de cet ensemble (exponentiel dans le pire des cas). Une représentation compacte offrant une complexité spatiale raisonnable dans certains cas « utiles » serait intéressante.

3.1 Représentation des solutions par contraintes complètes

Cette première approche consiste à calculer l'ensemble des solutions sous forme de contraintes complètes (CC). Ces contraintes sont des produits cartésiens de sous-domaines des variables. Leur intérêt est d'offrir une représentation compacte et de permettre une manipulation en compréhension.

Nous proposons deux méthodes [47] : un partitionnement par CC et un recouvrement partiel par CC maximales de l'ensemble des solutions.

1. la première méthode procède par décomposition de la micro-structure du CSP. Chaque contrainte est reformulée sous forme d'un ensemble de CC disjointes deux à deux, et chaque sous-CSP (résultant du choix d'une CC par contrainte initiale) possède pour ensemble de solutions une CC. L'ensemble de ces sous-problèmes fournit ainsi la partition désirée. L'existence d'un algorithme polynomial de décomposition de contrainte en CC, ainsi que le faible coût de composition des CC garantit une complexité théorique quasi-équivalente en pire cas à celle d'un calcul standard des solutions. D'un point de vue pratique, cette approche est profitable pour les CSP de grande taille et particulièrement pour les CSP à faible densité et contraintes lâches. Le calcul de toutes les solutions peut toutefois rester prohibitif.

2. dans ce cas, la seconde méthode [46] permet de générer une, voire plusieurs, CC de taille maximale. Son principe repose sur une recherche arborescente enrichie de techniques de filtrage. Le surcoût engendré par rapport au calcul d'une solution simple s'avère négligeable pour tout type de problème.

Selon la difficulté du problème, l'une ou l'autre des méthodes sera donc à privilégier.

3.2 Représentation de l'ensemble des solutions par automates

Une seconde approche, qui n'est pas à opposer à l'approche précédente, consiste à représenter l'ensemble des solutions par un automate d'états fini déterministe. Cette représentation a été proposée par Vempaty [67]. Elle s'inspire des travaux de Bryant [12] concernant la représentation des fonctions booléennes par des OBDD (Ordered Binary Decision Diagrams), cf. section 2.5.

Le principe repose sur la constatation suivante : en fixant un ordre sur les variables du CSP, l'ensemble des solutions apparaît comme un langage rationnel. Il peut donc être représenté par un automate déterministe. L'intérêt de cette approche réside dans la possibilité de récupérer les nombreux résultats issus du domaine des automates :

- l'automate déterministe minimal fournit une représentation compacte (et canonique) de l'ensemble des solutions d'un CSP ;
- la composition des automates déterministes permet la construction incrémentale de l'ensemble des solutions, ainsi que son maintien lors de l'ajout de nouvelles contraintes : il suffit de réaliser le produit de l'ancien automate avec celui associé à la contrainte ajoutée ;
- il est possible de prendre en compte le futur retrait d'une contrainte lors de son ajout. On ajoute à chaque contrainte une variable 0/1 exprimant la présence ou non de la contrainte. Le retrait s'effectuera en imposant une valeur à cette variable ;
- les algorithmes de minimisation et de composition d'automates déterministes ont une complexité faible (linéaire dans la taille de(s) automate(s)).

Cependant l'approche par automate déterministe pose des problèmes. Le principal réside dans le choix de l'ordre sur les variables du CSP. En effet, la taille de l'automate dépend fortement de cet ordre. De plus il existe des CSP pour lesquels tous les ordres conduisent à un automate de taille exponentielle en fonction du CSP.

Nous nous sommes intéressés à la recherche de bons ordres en étudiant les ordres classiques utilisés en CSP (largeur de Freuder, tree-width, band-width). Il apparaît que les ordres conduisant aux MDFA minimaux ne sont pas nécessairement ceux de largeur minimum. De plus ces largeurs ne permettent pas d'exprimer une borne pour la taille des MDFA minimaux (excepté pour le cas trivial où le graphe de contraintes est une chaîne) [2].

Aussi nous travaillons [2] sur une amélioration de cette représentation : les automates non-déterministes. Les raisons qui nous amènent à l'étude des automates non-déterministes sont :

- le non-déterminisme permet de diminuer la taille de la représentation (d'un facteur qui peut être exponentiel) ;
- les contraintes complètes (CC) peuvent être décrites comme des automates non-déterministes ;

– la construction incrémentale de l'ensemble des solutions ainsi que son maintien peuvent, comme pour les automates déterministes, être réalisés par composition d'automates.

Le problème de la minimisation d'automate non-déterministe bien que simplifié dans notre problématique (les langages considérés sont finis et composés de mots de même longueur) est un problème NP-difficile. En nous restreignant à la classe des automates non ambigus (entre deux états il n'existe pas deux chemins étiquetés par un même mot), ce problème reste ouvert.

On peut cependant définir des opérations de réduction d'automates qui, sans garantir la minimalité, permettent d'obtenir des « petits » automates non déterministes.

4 Relaxation automatique de contraintes

4.1 Prise en compte des préférences

Les préférences peuvent faire partie intégrante des connaissances caractérisant certains problèmes pratiques. Les approches classiques, qui doivent gommer ces préférences faute de pouvoir les représenter ou les intégrer, n'abordent pas directement ces problèmes. Cette forme de flexibilité des contraintes peut en fait être comprise soit comme des préférences entre affectations acceptables de variables de décision, soit des connaissances sur la relâchabilité de problèmes surcontraints.

En effet, certains problèmes, si on ne tient pas compte de ces préférences, ne sont *pas assez contraints* et le nombre de solutions peut être alors prohibitif. Par exemple, dans le projet *Serriste*, il peut exister de très nombreux réglages des équipements de la serre, à partir du moment où seraient négligés les critères de coût et de sécurité. En limitant la recherche aux solutions préférées, on va ramener le nombre de solutions à une quantité raisonnable. Les préférences apparaissent ici directement comme des critères permettant de guider la recherche vers les meilleures solutions.

Inversement, certains problèmes sont *surcontraints*, à savoir qu'ils n'ont aucune solution alors que l'utilisateur en veut une. La prise en compte des préférences permet de relâcher certaines contraintes et de donner ainsi des solutions (celles qui seront jugées « les moins mauvaises »). C'est typiquement le cas en ordonnancement, où une solution ne peut souvent être obtenue que moyennant, par exemple, une relaxation des délais des clients les moins importants.

De façon générale, la flexibilité permet d'*ordonner* (partiellement ou totalement) les solutions d'un problème. Remarquons que la résolution d'un CSP classique équivaut à partitionner l'ensemble des instanciations en deux classes : celles qui sont effectivement solutions du problème et celles qui ne le sont pas. L'introduction de préférences permet de raffiner cet ordre en créant des classes intermédiaires.

Le projet a particulièrement contribué à formaliser cette notion de contraintes flexibles. Le travail a principalement porté sur le modèle des Problèmes de Satisfaction de Contraintes Floues (FCSP), où chaque contrainte est définie non plus par l'ensemble des combinaisons de valeurs qui la satisfont, mais par un ensemble flou – l'ensemble des valeurs qui la satisfont *plus ou moins*. Dans ce cadre, une contrainte est satisfaite à un certain degré, plutôt que satisfaite ou non satisfaite. La richesse de ce modèle permet de prendre en compte à la fois des préférences entre valeurs et des priorités sur les contraintes.

Dans ce modèle, l'acceptabilité globale d'une solution devient elle aussi une notion graduelle : quand aucune affectation complète n'est totalement acceptable, les *meilleures* solutions sont celles qui maximisent le degré de satisfaction de la moins satisfaite des contraintes. Ce modèle, qui revient à un problème d'optimisation de type max-min, a été étudié (sous sa forme générale ou sous des formes plus spécifiques) dans [50, 58, 28] [1] . Un raffinement du critère d'optimalité max-min fondé sur l'utilisation d'un ordre lexicographique permettant de départager les solutions réalisables jugées équivalentes par le critère max-min a été étudié dans [27]. Une approche utilitariste (additive), où les degrés de satisfaction sont remplacés par des *pénalités*, est en cours (les bases logiques de cette approche ont été étudiées dans [25]).

Une application des FCSP aux problèmes d'ordonnancement de production de type « *job-shop* » est étudiée dans [30] ; cette formulation inclut l'existence de délais flexibles et la présence de tâches à durées mal connues.

Par ailleurs le travail a aussi consisté à proposer des algorithmes de filtrage et de recherche de solution associés à ces nouveaux formalismes de satisfaction de contraintes flexibles : les problèmes sont résolus en généralisant aux FCSP les algorithmes issus des CSP (propagation de contraintes, recherche de solutions). Il est intéressant de noter que la complexité dans le pire des cas reste du même ordre que pour la satisfaction de contraintes classiques (une étude de la complexité des FCSP figure dans [15]). Ces algorithmes ont été implémentés et testés sur des instances de FCSP issus de problèmes d'ordonnancement générées aléatoirement [29].

4.2 Prise en compte des données mal connues

Dans certains cas, la description du problème elle-même est incomplète, soit à cause de données mal connues, c'est-à dire que la valeur des paramètres (non contrôlables) du problème est imprécise ou vague (« la longueur de tel composant est d'environ 5 cm »), soit à cause d'incertitudes sur l'énoncé du problème lui-même, le plus souvent parce que la réalisation d'un événement ne peut pas être prévue avec certitude dans le cas d'une résolution prévisionnelle (pannes dans *Hermes*, prévisions météo dans *Sacogit*, etc.).

Pour représenter ces deux types d'incomplétude, il est possible d'associer aux données du problème des mesures d'incertitude, qui peuvent par exemple être des probabilités (quand la nature de ces données l'autorise, c'est-à-dire quand l'incertitude est de type « hasard »), ou des possibilités/nécessités, qui permettent une mesure plus prudente, plus qualitative, lorsque l'incertitude est plutôt de type « ignorance » (sans interprétation statistique).

Les FCSP étant fondés sur la théorie des possibilités, une hypothèse de commensurabilité satisfaction/incertitude a permis d'intégrer au modèle FCSP des contraintes entre des paramètres mal connus, incontrôlables, et dont les valeurs possibles sont restreintes par une distribution de possibilité, et des variables de décision, contrôlables. Il s'agit de rechercher

1. Les approches antérieures pour la satisfaction de contraintes flexibles, incluent [61] qui utilisent une approche égalitariste similaire aux FCSP, [24] (approche de type lexicographique), [10] et [35] qui étudie la relaxation de problèmes dans un cadre général. [33] s'intéressent à maximiser le nombre de contraintes satisfaites.

une affectation des variables de décision qui sera solution du problème réel quelles que soient les valeurs effectives des paramètres incontrôlables : le degré de satisfaction d'une contrainte, éventuellement flexible, entre paramètres incontrôlables et variables de décision est alors le degré de nécessité que la contrainte soit satisfaite. De la même façon, et moyennant des hypothèses d'indépendance et de commensurabilité, il a été possible de prendre en compte des contraintes dont on n'est sûr qu'à un certain degré qu'elles appartiennent au problème réel; il s'agit encore de rechercher une solution qui maximise la certitude que le problème réel sera satisfait. Ce travail [28] constitue un premier pas vers l'introduction des deux types de valuations (préférences et incertitudes) dans les CSP.

Il est a priori moins naturel d'intégrer directement des préférences de nature qualitative avec de l'incertitude probabiliste, du fait de l'hétérogénéité des modèles de référence. [26] et [51] proposent un traitement des incertitudes probabilistes sur l'énoncé du problème, se limitant à des problèmes ne mettant pas de préférences en jeu. Sous l'hypothèse que les événements incertains qui contrôlent l'existence des contraintes soient mutuellement indépendants, on associe à chaque contrainte potentielle la probabilité qu'elle soit effectivement une contrainte du problème réel [26]. On recherche alors une solution qui maximise cette probabilité. Une extension à des familles indépendantes de contraintes est proposée par [51]. La prise en compte de préférences dans ces modèles est l'objet de travaux en cours ; une étude préliminaire figure dans [31], ainsi qu'une application à la planification agricole.

5 Traitement de problèmes hiérarchiques

Dans de nombreux problèmes pratiques, une limitation importante des CSP reste la nécessité d'exprimer un ensemble de variables défini statiquement avec des contraintes portant sur un ensemble de variables fixe lui-aussi.

Les problèmes de conception sont souvent de nature hiérarchique : un sous-problème (variables et contraintes) n'apparaît que lorsque certains choix de plus haut niveau ont été faits.

De même, dans les problèmes de configuration, le nombre de composants d'un type donné est lui-même une inconnue du problème et des contraintes peuvent porter sur un nombre de variables non déterminé dans l'énoncé du problème.

Des extensions au formalisme des CSP sont donc nécessaires pour exprimer et traiter ces problèmes.

5.1 Métacontraintes

Pour essayer de répondre au problème d'expression de hiérarchies, nous avons étendu le formalisme des problèmes de satisfaction de contraintes dynamiques. Ce formalisme reprend le formalisme de base des CSP et remplace certains de ses objets par des variables. En particulier, l'ensemble des variables sur lesquelles porte une contrainte et la relation qu'elle représente, qui sont des données d'un CSP traditionnel, deviennent des variables dans un problème de contrainte dynamique. Ceci autorise l'expression de *métacontraintes* qui sont des contraintes sur la structure d'autres contraintes. Alors que l'interprétation de contraintes classiques a pour but d'attribuer une valeur aux variables du problème, l'interprétation de métacontraintes construit ou ajuste la structure du problème lui-même, permettant la reconfiguration du problème au cours de sa résolution. Dans [7] et [6], nous avons montré l'utilisation des métacontraintes dans le cadre du maintien de solution. Ce formalisme permet d'exprimer des relations comme « *le coût d'un objet est la somme du coût de ses composants* », sans connaître *a priori* quels seront les composants qui constitueront la solution.

5.2 CSP à variables conditionnelles

Une autre approche étendant le formalisme CSP par des variables possédant des conditions de présence a été définie par Mittal et Falkenheiner [53]. Ils ont proposé un algorithme de satisfaction de contraintes qui introduit au cours de la procédure d'énumération avec retour-arrière de nouvelles variables quand les conditions de présence de ces variables sont vérifiées.

Nous avons alors cherché à rendre cette approche « dynamique » (au sens du maintien de solution). Il s'agit alors de trouver une solution proche d'une solution initiale rendue invalide après un ajout de contrainte. La proximité s'exprime ici par la structure de la solution : une solution sera proche si elle est formée des mêmes variables ou d'un ensemble de variables peu différent de la solution initiale. L'algorithme de réparation heuristique de minimisation des conflits [52] a ainsi été étendu aux CSP à variables conditionnelles : l'heuristique de choix de la variable à réparer prend en compte les conflits de structure en plus des traditionnels conflits de valeur.

6 Exploitation de la nature des contraintes, des domaines

Dans les années 80, de nombreux travaux ont porté sur l'exploitation de l'agencement des contraintes entre elles (structure du graphe de contraintes). Ce n'est que depuis peu que la nature des contraintes est prise en compte pour développer des algorithmes efficaces, ou dégager de nouvelles classes polynomiales de réseaux de contraintes [55, 64, 66, 44, 65]. Cet axe de recherche a également été développé parmi les membres du groupe. Des résultats intéressants ont été obtenus sur les contraintes fonctionnelles (partie 6.1), les contraintes « 0/1/All » (partie 6.2), les contraintes « tous différents » [57], les contraintes commutatives (partie 6.3), et les contraintes impliquant des valeurs substituables (partie 6.4).

Par ailleurs, de nombreuses applications en IA nécessitent l'extension du modèle CSP pour accepter plus que des contraintes portant sur de simples ensembles finis de valeurs. En raisonnement temporel par exemple, notons l'extension aux intervalles de temps [23]. D'autres extensions peuvent s'avérer nécessaires ou utiles pour d'autres types de problèmes. En particulier, dans les systèmes orientés objets, on peut prendre en compte le caractère hiérarchique des domaines pour simplifier le codage des contraintes et améliorer les algorithmes (partie 6.4). Dans les systèmes modélisant le monde physique, il est souvent nécessaire d'utiliser des variables à domaines continus soumises à des contraintes numériques (partie 6.5). De même, dans les problèmes d'aménagement spatial, il faut manipuler des domaines et des contraintes d'un type particulier pour représenter des informations géométriques (partie 6.6).

6.1 Les contraintes fonctionnelles

Une contrainte binaire liant les variables x_i et x_j est dite *fonctionnelle*, et notée $x_i \rightarrow x_j$ si et seulement si toute valeur du domaine de x_i (noté d_i) possède au plus une valeur compatible dans le domaine de x_j (d_j) pour cette contrainte (en d'autres termes, ssi sa relation représente une fonction partielle de d_i vers d_j) [16]. La contribution essentielle est l'obtention d'une nouvelle classe polynomiale pour le problème de satisfiabilité dans le cas où le CSP contient un ensemble de contraintes fonctionnelles « bien agencées ».

Une nouvelle consistance locale tirant parti du caractère fonctionnel des contraintes est définie : la *consistance de pivot* ; un filtrage permettant de l'obtenir est également proposé [17]. On détermine ensuite un ensemble $\mathcal{R}$ de variables puis un ordre associé (appelé $\mathcal{R}$-compatible), permettant d'obtenir le résultat suivant :

Théorème *Soit un CSP $\mathcal{P} = (X, D, C)$, un sous-ensemble de ses variables $\mathcal{R} \subset X$, et un ordre $\mathcal{R}$-compatible. Si $\mathcal{P}$ vérifie la consistance de pivot, alors toute instanciation consistante de $\mathcal{R}$ peut se prolonger linéairement en une solution du CSP.*

La résolution du CSP initial est donc ramenée à celle du sous-problème $\mathcal{P}_{\mathcal{R}}$ induit par $\mathcal{R}$ (*i.e.* les variables sont celles de $\mathcal{R}$ et les contraintes celles liant deux variables de $\mathcal{R}$).

Un ensemble $\mathcal{R}$ de cardinalité minimale ainsi que l'ordre $\mathcal{R}$-compatible associés peuvent s'obtenir en $O(e + n)$; l'exécution du filtrage par consistance de pivot nécessite un temps de $O((n^2 - r^2).d^2)$ si r est la cardinalité de $\mathcal{R}$; enfin, la résolution du sous-problème $\mathcal{P}_{\mathcal{R}}$ requiert $O(e_{\mathcal{R}} d^r)$, où $e_{\mathcal{R}}$ est le nombre de contraintes de $\mathcal{P}_{\mathcal{R}}$ (cf. section 1 pour la définition des paramètres e, n, d et r).

6.2 Les contraintes « 0/1/All »

Nous avons identifié une classe de contraintes binaires pour laquelle le problème de satisfaction de contraintes devient facile [14] : lorsque pour chaque paire de variables $\{x_i, x_j\}$, chaque valeur possible pour x_i est reliée à zéro, une ou toutes les valeurs possibles pour x_y (0/1/All), il existe un algorithme de complexité de l'ordre de n^3 pour trouver une solution. Nous avons démontré que toute autre classe de contraintes fermée par permutation et restriction sur les valeurs des domaines est NP-complète pour la satisfiabilité.

6.3 Bidirectionnalité et commutativité

La consistance d'arc a toujours été un des principaux centres d'intérêt dans l'étude des CSP. De nombreux algorithmes ont été proposés pour l'obtenir. Mais, étrangement, aucun des algorithmes existant ne tirait parti de la propriété de bidirectionnalité (i.e. $R_{ij}(a, b)$=$R_{ji}(b, a)$) qui est pourtant vérifiée par toutes les contraintes. En utilisant cette propriété, il a été proposé un algorithme efficace inspiré d'AC-6. Il a ensuite été étendu en un algorithme optimal en terme de nombre de tests de consistance effectués [8].

En se basant sur ce principe de l'inférence de nouveaux tests de consistance à partir de tests déjà effectués, il est possible de produire des algorithmes qui traitent efficacement des contraintes vérifiant certaines propriétés : réflexives (*i.e.* $R_{ij}(a, a) = vrai$), anti-réflexives (i.e. $R_{ij}(a,a) = faux$), commutatives (i.e. $R_{ij}(a,b) = R_{ij}(b,a)$), anti-commutatives (i.e. $R_{ij}(a,b) = \neg R_{ij}(b,a)$.

6.4 CSP à domaines hiérarchiques et substituabilité

Nous allons exploiter des ordres partiels sur les domaines, selon deux approches complémentaires : l'une dans le cadre des systèmes orientés objets, où nous utilisons des connaissances *a priori* sur la structuration des domaines, et l'autre dans le cadre général des CSP où la connaissance est acquise.

Première approche – Notre première motivation repose sur l'utilisation des objets des langages orientés objets comme valeurs possibles de variables. La signification intuitive de la relation de spécialisation/généralisation « est-un » nous amène à poser l'hypothèse suivante : si un objet, valeur d'une variable contrainte, satisfait une contrainte R_{ij}, alors toutes les spécialisations de cet objet satisfont aussi cette contrainte. Et à l'inverse, s'il ne satisfait pas la contrainte R_{ij}, alors aucune de ses généralisations ne la satisfait.

En s'appuyant sur cette hypothèse, nous pouvons définir une classe spéciale de CSP (les DHCSP : CSP à Domaines Hiérarchiques) où un ordre partiel est défini sur chaque domaine et où toute contrainte vérifie l'hypothèse. La prise en compte de cette structure hiérarchique des domaines peut avoir plusieurs avantages :

- elle peut permettre de simplifier la définition des contraintes données en extension.
- les algorithmes de filtrage et la résolution globale peuvent être améliorés. Nous avons proposé une spécialisation (HAC-6) de l'algorithme AC-6 de filtrage par consistance d'arc [45].
- dans un CSP où les domaines sont hiérarchiques, une solution trouvée peut représenter un ensemble de solutions. Une idée naturelle est alors de chercher uniquement les solutions « maximales » au sens de l'ordre produit induit par les ordres partiels sur les domaines.

Seconde approche – D'une manière complémentaire, nous allons chercher à calculer des ordres définis selon la sémantique de contraintes ordinaires.

Notre point de départ est la notion d'interchangeabilité entre valeurs définie par Freuder [36]. Nous utilisons une forme plus faible de cette relation, appelée substituabilité [36].

Pour un CSP binaire $\mathcal{P}$, et pour deux valeurs a et b du domaine d_i, a est substituable par b ssi toute solution où figure a reste solution si on remplace a par b. Notons que deux valeurs a et b sont interchangeables si a est substituable par b et b substituable par a.

Ces notions n'étant pas directement utilisables, car elles nécessitent de connaître l'ensemble des solutions, nous utilisons des conditions locales suffisantes : la voisinage-interchangeabilité et la voisinage-substituabilité [36].

Soit a et b, deux valeurs du domaine Di. On dit que a est voisinage-substituable par b ssi l'ensemble des supports de a est inclus dans celui de b. Elles sont voisinage-interchangeables si elles sont mutuellement voisinage-substituables.

La relation de voisinage-substituabilité définit un pré-ordre partiel sur chacun des domaines d'un CSP. Ces pré-ordres partiels, appelés NS-ordres (Neighbourhood Substituability), ont pour principal intérêt le fait que dans chaque domaine du CSP, on peut supprimer les valeurs non-maximales sans perdre la satisfiabilité du CSP.

Nous avons exploité cette propriété dans deux directions :

- nous avons proposé un algorithme d'une structure semblable à celle d'AC-6, qui réduit le problème de manière à ce qu'il ne comporte plus que des valeurs maximales pour les NS-ordres : le problème obtenu a la même satisfiabilité que le problème initial [4].
- nous avons généralisé la voisinage-substituabilité en k-voisinage-substituabilité, à la manière de la k-consistance.

Outre une complexité théorique satisfaisante, nous avons montré l'utilité de notre démarche dans la résolution de certaines catégories de problèmes.

6.5 Domaines numériques (discrets ou continus)

Les modélisations du monde physique font fréquemment intervenir des systèmes de contraintes numériques et des données imprécises ; dans ces modélisations le problème se ramène souvent à déterminer les domaines de variation des variables. Malheureusement, lorsque les contraintes sont non linéaires, il n'existe pas de méthode générale pour calculer les domaines de variation. Il est cependant possible d'obtenir une approximation de ces domaines de variation grâce à des algorithmes de filtrage analogues à ceux utilisés sur les CSP. Par exemple, l'algorithme de propagation d'intervalles [18] est une adaptation de l'algorithme de Waltz qui prend en compte les particularités des contraintes numériques. De la même façon, les algorithmes de *k-B*-consistance [48, 49] sont une adaptation aux contraintes numériques des algorithmes de *k*-consistance [32].

Dans [49] nous avons étudié une évolution de l'algorithme de propagation d'intervalles visant à exploiter son comportement dynamique. Il faut savoir que l'une des principales limitations de l'algorithme de propagation d'intervalles réside dans l'existence de convergences lentes se traduisant sur certains systèmes de contraintes par des temps de réponse prohibitifs. Nous avons tout d'abord établi un lien entre les convergences lentes et l'existence de phénomènes cycliques. Un cycle de propagation est défini par une liste ordonnée de contraintes, par exemple (a, b, c, d, e). L'exécution de l'algorithme de propagation d'intervalles au cours du phénomène de convergence lente (et après une phase transitoire) est équivalente à n évaluations du cycle, noté $(a, b, c, d, e)^n$. L'approche consiste à détecter des phénomènes de convergence lente durant l'exécution de l'algorithme de propagation d'intervalles, puis à utiliser leur caractère cyclique pour améliorer les performances. À cette fin, l'algorithme proposé procède à deux types de simplification de cycle :

- dans certains cycles de propagation, il peut exister des contraintes qui sont évaluées inutilement (les réductions de domaines qu'elles entraînent sont subsumées par d'autres contraintes du cycle). Si la contrainte b est une telle contrainte inutile, alors $(a, b, c, d, e)^n$ est équivalent à $(a, c, d, e)^n$;

– une deuxième type de simplification de cycle consiste à détecter les contraintes du cycle de propagation qui peuvent être différées. Par exemple, si les contraintes d et e sont différables, alors $(a, c, d, e)^n$ est équivalent à $(a, c)^n(d, e)$.

Ces simplifications, outre une amélioration notable du temps de calcul, ont aussi une incidence sur la complexité théorique de l'algorithme de propagation d'intervalles, qui est fonction linéaire du nombre de contraintes apparaissant dans le cycle de propagation.

6.6 Les contraintes géométriques

Nous avons développé le prototype d'un environnement d'aide à l'aménagement spatial. Les objets manipulés sont des rectangles isothétiques – i.e. dont les côtés sont parallèles aux axes – de dimensions variables dans le plan.

Au cours de nos travaux, une nouvelle forme de consistance adaptée aux problèmes d'aménagement spatial, la consistance d'arc semi-géométrique, a été définie. Un retour arrière de type FOF [34] sur des contraintes n-aires a été également développé. La validation de l'extension des techniques CSP a principalement porté sur des exemples concrets de plans de sol.

Nos recherches font apparaître que l'efficacité d'une approche basée sur les contraintes pour résoudre les problèmes d'aménagement spatial requiert principalement deux éléments. Premièrement, la consistance du graphe lors de la recherche des solutions doit être d'un degré au moins égal à celui de la consistance d'arc semi-géométrique [13]. Ceci implique d'une part une modélisation multi-dimensionnelle pour représenter les domaines des variables de position et d'autre part l'existence d'une méthode de filtrage pour chaque contrainte géométrique. Deuxièmement, l'introduction de contraintes redondantes telles que la contrainte cumulative [1] est quasi indispensable.

7 Conclusion

Cet exposé rapide montre le foisonnement actuel que l'on peut observer autour du formalisme CSP au niveau national et international ainsi que l'importante activité de la communauté française dans le domaine. Depuis les premiers travaux de Montanari [54], le cadre CSP s'est énormément enrichi et la quantité des algorithmes et extensions existants est maintenant très importante. Parallèlement, l'utilisation des techniques CSP pour le traitement de problèmes réels a, relativement, peu évolué et la grande majorité des outils s'appuyant sur les techniques CSP continuent d'utiliser des algorithmes tels que le filtrage via AC3, le « *Real Full LookAhead* » ou le « *Forward Checking* ».

En fait, il reste souvent un travail important pour passer de la proposition d'un algorithme de traitement de CSP *binaires*, dont les contraintes sont définies *en extension* à son application à des problèmes réels, qui sont souvent de grande taille et dans lesquels des contraintes non binaires sont généralement présentes.

Surtout, l'efficacité et la robustesse des techniques proposées doivent être établies plus fortement que par la simple utilisation de CSP binaires aléatoires (dont le modèle de génération actuel reste encore très primitif). S'il est difficile de demander à chaque chercheur de traiter différents problèmes réels en sus des traditionnels CSP aléatoires, deux directions, orthogonales aux traditionnelles directions de recherche «scientifiques» devraient permettre d'améliorer la situation actuelle :

– établissement d'une collection de « *benchmarks* » issus de problèmes réels et traduits dans une syntaxe proche du modèle CSP traditionnel (qu'il reste à définir) ;

– construction d'outils avancés, à vocation expérimentale plus que commerciale. La variété des algorithmes, la facilité d'extension, l'ouverture devraient être privilégiées, éventuellement au prix d'une efficacité moindre. Encore une fois, la programmation linéaire a pu montrer comment, une fois l'intérêt d'une technique clairement établi dans des outils expérimentaux, celle-ci est rapidement intégrée dans les outils commerciaux.

Références

[1] Aggoun (A.) et Beldiceanu (N.). – Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, vol. 17, n° 7, 1993, pp. 57–73.

[2] Amilhastre (J.). – *Représentation par automates de l'ensemble des solutions d'un Problème de Satisfaction de Contraintes*. – Rapport technique n° 94056, LIRMM, 1994.

[3] Bellicha (A.). – Maintenance of Solution in a Dynamic Constraint Satisfaction Problem. *In : Proc. of Applications of Artificial Intelligence in Engineering VIII*, pp. 261–274. – Toulouse, France, 1993.

[4] Bellicha (Amit), Habib (Michel), Vilarem (Marie Catherine), Capelle (Christian) et Kökény (Tibor). – CSP techniques using partial orders on domain values. *In : Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*. – Amsterdam, août 1994.

[5] Berlandier (P.) et Neveu (B.). – Maintaining Arc Consistency through Constraint Retraction. *In : Proc. of the 6^{th} IEEE International Conference on Tools with Artificial Intelligence (TAI94)*. – New Orleans, LA, 1994.

[6] Berlandier (P.) et Trousse (B.). – Expression et interprétation de métacontraintes pour le maintien de solution en conception. *In : Proc. of the 13^{th} international Avignon workshop*. – Avignon, 1993.

[7] Berlandier (Pierre). – The use and interpretation of meta level constraints. *In : Proc. EPIA*. pp. 271–280. – Springer–Verlag.

[8] Bessière (C.) et Régin (J.C.). – An arc-consistency algorithm optimal in the number of constraint checks. *In : Proceedings 6th IEEE Conference on Tools with AI*. – New-Orleans, LA, 1994.

[9] Bessière (Christian). – Arc-consistency in dynamic constraint satisfaction problems. *In : Proc. of AAAI-91*, pp. 221–226. – Anaheim, CA, 1991.

[10] Borning (A.), Mahert (M.), Martindale (A.) et Wilson (M.). – Constraint hierarchies and logic programming. *In : Int. conf. on logic programming*, pp. 149–164.

[11] Bouquet (Fabrice). – *Complexité des Diagrammes de Décision Binaires pour la Résolution de Problèmes Dynamiques*. – Mémoire de dea, Marseille, France, Université de Provence – Laboratoire d'Informatique de Marseille, juillet 1994.

[12] Bryant (R. E.). – Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, vol. 24, n° 3, septembre 1992, pp. 293–318.

[13] Charman (P.). – A constraint-based approach for the generation of floor plans. *In : Proceedings Workshop AAAI'94 on Spatial and Temporal Reasoning*. – Seattle, WA, 1994.

[14] Cooper (Martin C.). – Characterising tractable constraints. *Artificial Intelligence*, vol. 65, 1994, pp. 347–361.

[15] Cooper (Martin C.). – On the complexity of fuzzy constraint satisfaction problems, 1994. École d'été sur les problèmes de satisfaction de contraintes.

[16] David (Philippe). – When functional and bijective constraints make a CSP polynomial. *In : Proc. of the 13th IJCAI*. – Chambery, France, août 1993.

[17] David (Philippe). – Consistance de pivot et CSP fonctionnels – Une consistance partielle pour des solutions globales. *Revue Francaise d'Intelligence Artificielle*, vol. 8, n° 2, septembre 1994, pp. 145–185.

[18] Davis (E.). – Constraint propagation with interval labels. *Artificial Intelligence*, vol. 32, n° 3, juillet 1987, pp. 281–331.

[19] de Kleer (J.). – A comparison of ATMS and CSP techniques. *In : Proc. of the 11th IJCAI*, pp. 290–296. – Detroit, MI, août 1989.

[20] Debruyne (R.). – *DnAC6*. – Mémoire de DEA n° RR 94054, LIRMM, 1994.

[21] Dechter (Rina). – Enhancement schemes for constraint processing : Backjumping, learning and cutset decomposition. *Artificial Intelligence*, vol. 41, n° 3, 1990, pp. 273–312.

[22] Dechter (Rina) et Dechter (Avi). – Belief maintenance in dynamic constraint networks. *In : Proc. of AAAI-88*, pp. 37–42. – St. Paul, MN, 1988.

[23] Dechter (Rina), Meiri (Itay) et Pearl (Judea). – Temporal constraint networks. *Artificial Intelligence*, vol. 49, 1991, pp. 61–95.

[24] Descotte (Y.) et Latombe (J.C.). – Making compromises among antagonist constraints in a planner. *Artificial Intelligence*, vol. 27, n° 2, novembre 1985.

[25] Dupin de Saint Cyr (Florence), Lang (Jérôme) et Schiex (Thomas). – Gestion de l'inconsistance dans les bases de connaissances : une approche syntaxique basée sur la logique des pénalités. *In : Actes de RFIA'94*, pp. 507–518. – France, 1994.

[26] Fargier (H.) et Lang (J.). – Uncertainty in constraint satisfaction problems: a probabilistic approach. *In : Proc. of ECSQARU '93, LNCS 747*, pp. 97–104. – Grenade, Spain, novembre 1993.

[27] Fargier (H.), Lang (J.) et Schiex (T.). – Selecting preferred solutions in Fuzzy Constraint Satisfaction Problems. *In : Proc. of the 1st European Congress on Fuzzy and Intelligent Technologies*.

[28] Fargier (Hélène). – *Problèmes de satisfaction de contraintes flexibles et application à l'ordonnancement de production*. – Toulouse, France, Thèse de PhD, Université Paul Sabatier – Institut de Recherche en Informatique de Toulouse, juin 1994.

[29] Fargier (Hélène). – Problèmes d'ordonnancement flexibles: principes et expérimentations. *In : 4èmes Journées Nationales sur les Ensembles Flous et leurs Applications*. – Lille, novembre 1994.

[30] Fargier (Hélène). – The use of fuzzy constraints in job-shop scheduling. *In : Proc. of the $2^n d$ World Congress on Expert Systems*.

[31] Fargier (Helene), Lang (Jerome), Martin-Clouaire (Roger) et Rellier (Jean-Pierre). – Uncertainty and flexibility in constraint satisfaction: A case study. *In : Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*, éd. par Schiex (Thomas) et Bessière (Christian). – Amsterdam, août 1994.

[32] Freuder (E. C.). – Synthesizing constraint expressions. *Communications of the ACM*, vol. 21, novembre 1978, pp. 958–966.

[33] Freuder (E.C.) et Wallace (R.J.). – Partial constraint satisfaction. *Artificial Intelligence*, vol. 58, décembre 1992, pp. 21–70.

[34] Freuder (Eugene) et Hubbe (Paul). – Extracting constraint satisfaction subproblems. *In : Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*, éd. par Schiex (Thomas) et Bessière (Christian). – Amsterdam, août 1994.

[35] Freuder (Eugene C.). – Partial constraint satisfaction. *In : Proc. of the 11^{th} IJCAI*, pp. 278–283. – Detroit, MI, 1989.

[36] Freuder (Eugene C.). – Completable representations constraint satisfaction problems. *In : Proc. of the KR'91 Conference*, pp. 186–195. – Cambridge MA, 1991.

[37] Frost (Daniel) et Dechter (Rina). – Dead-end driven learning. *In : Proc. of AAAI-94*. – Seattle, WA, août 1994.

[38] Gent (I.) et Walsh (T.). – An Empirical Analysis of Search in GSAT. *Journal of Artificial Intelligence Research*, vol. 1, 1993, pp. 47–59.

[39] Ghedira (Khaled). – *MASC : une approche multi-agent des problèmes de satisfaction de contraintes*. – Toulouse, France, Thèse de doctorat, École Nationale Supérieure de l'Aéronautique et de l'Espace (ENSAE), juin 1993.

[40] Ghedira (Khaled). – Distributed simulated re-annealing for dynamic constraint satisfaction problems. *In : Proceeding of the 6^{th} IEEE International Conference on Tools with Artificial Intelligence*. – New Orleans, USA, 1994.

[41] Ghedira (Khaled). – Partial constraint satisfaction by a multi-agent approach combined with simulated annealing process. *In : Proc. of the 14^{th} international Avignon workshop*. – Paris, France, 1994.

[42] Ginsberg (Matthew L.). – Dynamic backtracking. *Journal of Artificial Intelligence Research*, vol. 1, 1993, pp. 25–46.

[43] Jégou (Philippe). – A logical approach to solve dynamic CSPs: Preliminary report. *In : Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*, pp. 87–94. – Amsterdam, The Netherlands, août 1994.

[44] Kirousis (Lefteris M.). – Fast parallel constraint satisfaction. *Artificial Intelligence*, vol. 64, 1993, pp. 147–160.

[45] Kökény (Tibor). – A new arc-consistency algorithm for CSPs with hierarchical domains. *In : Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*, éd. par Schiex (Thomas) et Bessière (Christian). – Amsterdam, août 1994.

[46] Lesaint (D.). – Maximal sets of solutions for constraint satisfaction problems. *In : ECAI'94*, pp. 110–114. – Amsterdam, 1994.

[47] Lesaint (David). – Specific solution sets for Constraint Satisfaction Problems. *In : Proc. of the 13th international Avignon workshop*.

[48] Lhomme (O.). – Consistency techniques for numeric CSPs. *In : Proc. of the 13th IJCAI*, pp. 232–238. – Chambery, France, août 1993.

[49] Lhomme (Olivier). – *Contribution à la résolution de contraintes sur les réels par propagation d'intervalles*. – Sophia Antipolis, Thèse de doctorat, Université de Nice Sophia Antipolis, juin 1994.

[50] Martin-Clouaire (R.). – Dealing with soft constraints in a constraint satisfaction problem. *In : Proc. of the International Conference on Information Processing of Uncertainty in Knowledge based Systems*, pp. 37–40. – Palma de Mallorca, juillet 1992.

[51] Martin-Clouaire (R.) et Rellier (J-P.). – Crop management planning as a CSP with advanced features. *In : Proc. of the IJCAI-93 Workshop on AI in Agriculture, Natural Resources and Environmental Sciences*, pp. 129–136. – Chambery, France, août 1993.

[52] Minton (S.), Johnston (M. D.), Philips (A. B.) et Laird (P.). – Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. *In : Proc. of AAAI-90*, pp. 17–24. – Boston, MA, 1990.

[53] Mittal (S.) et Falkenhainer (B.). – Dynamic constraint satisfaction problems. *In : Proc. of AAAI-90*, pp. 25–32. – Boston, MA, 1990.

[54] Montanari (U.). – Network of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.*, vol. 7, 1974, pp. 95–132.

[55] Perlin (Mark). – Arc consistency for factorable relations. *Artificial Intelligence*, vol. 53, 1992, pp. 329–342.

[56] Projet BAHIA. – étude comparative de trois formalismes en calcul propositionnel. *In : Actes des journées nationales du PRC GDR Intelligence Artificielle*, pp. 239–314. – Marseille, France, octobre 1992.

[57] Régin (J.C.). – A filtering algorithm for constraints of difference in CSPs. *In : Proc. of AAAI-94*, pp. 362–367. – Seattle, WA, 1994.

[58] Schiex (T.). – Possibilistic constraint satisfaction problems or "How to handle soft constraints ?". *In : Proc. of the 8^{th} Int. Conf. on Uncertainty in Artificial Intelligence*. – Stanford, CA, juillet 1992.

[59] Schiex (T.) et Verfaillie (G.). – Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools*, vol. 3, n° 2, 1994, pp. 187–207.

[60] Schiex (Thomas) et Verfaillie (Gérard). – Stubborness: an enhancement scheme for backjumping and nogood recording. *In : Proc. of the 12^{th} ECAI*, pp. 165–169. – Amsterdam, Netherlands, août 1994.

[61] Snow (P.) et Freuder (E.C.). – Improved relaxation and search methods for approximate constraint satisfaction with a maximin criterion. *In : Proc. of the 8^{th} biennal conf. of the canadian society for comput. studies of intelligence*, pp. 227–230.

[62] Trombettoni (G.). – CCMA*: a semi-Complete Constraint Maintenance Algorithm using constraint programming. *In : Proc. of the International Workshop at CSAM'93*, éd. par Meyer (M.). DFKI. – St. Petersburg, Russia, 1993. DFKI Research Report 93-39.

[63] Trombettoni (G.). – A constraint maintenance algorithm based on value propagation. *In : Poster in the 6th Portuguese Conference on Artificial Intelligence, EPIA'93*, éd. par Filgueras (Miguel) et Damas (Luis), p. 360. – Porto, Portugal, 1993. Lecture Notes in Artificial Intelligence 727.

[64] van Beek (Peter). – Reasoning about qualitative temporal information. *Artificial Intelligence*, vol. 58, décembre 1992, pp. 297–326.

[65] van Beek (Peter) et Dechter (Rina). – Constraint tightness versus global consistency. *In : Proceedings KR'94*. – Bonn, Germany, 1994.

[66] van Hentenryck (P.), Deville (Y.) et Teng (C.). – A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, vol. 57, 1992, pp. 291–321.

[67] Vempaty (Nageshwara Rao). – Solving constraint satisfaction problems using finite state automata. *In : Proc. of AAAI-92*, pp. 453–458. – San Jose, CA, 1992.

[68] Verfaillie (G.) et Schiex (T.). – Solution reuse in dynamic constraint satisfaction problems. *In : Proc. of AAAI-94*, pp. 307–312. – Seattle, WA, août 1994.

[69] Verfaillie (Gerard) et Schiex (Thomas). – Dynamic backtracking for dynamic CSPs. *In : Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*, éd. par Schiex (Thomas) et Bessière (Christian). – Amsterdam, août 1994.