Une comparaison de logiciels d'optimisation sur une large collection de modèles graphiques

D. Allouche[1] **S. de Givry**[1] **B. Hurley**[2] **G. Katsirelos**[1] **B. O'Sullivan**[2] **T. Schiex**[1]

[1] MIAT, UR-875, INRA, F-31320 Castanet Tolosan, France
[2] Insight Centre for Data Analytics, University College Cork, Ireland
simon.degivry@toulouse.inra.fr b.hurley@4c.ucc.ie

Résumé

Le cadre des modèles graphiques à *variables discrètes* permet de modéliser des problèmes d'optimisation NP-difficiles pour lesquels la fonction objectif se factorise en un ensemble de *fonctions locales*. L'interprétation graphique de ces modèles est que chaque fonction est représentée par une clique sur les variables de sa portée. Les modèles graphiques dits *déterministes* ont pour objectif de minimiser la somme des fonctions locales, pouvant aussi être des contraintes si seuls les coûts zéro ou infini sont utilisés. Les modèles graphiques dits *probabilistes* ont pour objectif de maximiser le produit des fonctions (des contraintes si usage uniquement des probabilités zéro ou un). Une transformation directe existe entre les deux classes de modèles graphiques, qui peuvent également être modélisés en programmation linéaire en nombres entiers ou en problème de satisfiabilité maximum en logique propositionnelle.

Dans cet article, nous évaluons plusieurs logiciels implémentant l'état de l'art en méthodes complètes d'optimisation sur une large collection de modèles graphiques déterministes et probabilistes issus de diverses compétitions Max-CSP 2008, Probabilistic Inference Challenge 2011, Weighted Partial Max-SAT Evaluation 2013, MiniZinc Challenge 2012 & 2013, ainsi que des collections provenant des problème de satisfaction de contraintes pondérées et en traitement d'images. Au total, 3018 instances sont mises à disposition dans cinq formats et sept formulations avec les scripts de conversion à `http://genoweb.toulouse.inra.fr/~degivry/evalgm`. Les résultats montrent que différents logiciels généralistes obtiennent de bons résultats sur plusieurs catégories de modèles graphiques, suggérant des opportunités pour une approche portfolio d'algorithmes.

1 Introduction

Les modèles graphiques permettent de représenter une distribution multivariée de manière compacte en exploitant la factorisation de la distribution par des fonctions locales. Nous nous restreignons au cas de variables discrètes. Ce cadre fait habituellement les hypothèses restrictives suivantes : faible arité des fonctions et taille restreinte des domaines permettant d'exprimer des fonctions quelconques en extension.

En intelligence artificielle, le problème de satisfaction de contraintes (CSP) et sa variante pondérée pour l'optimisation (WCSP) consiste à trouver une affectation de toutes les variables qui minimise une distribution définie par la somme de fonctions locales, les contraintes étant capturées par des fonctions à valeurs dans $\{0, \infty\}$. Restreint à des variables Booléennes et le langage de la logique propositionnelle en forme normale conjonctive, le problème de satisfiabilité maximum (Max-SAT) a le même objectif. La programmation par contraintes (PPC) peut également modéliser ces problèmes en introduisant des variables de coût [27].

Les modèles graphiques probabilistes (PGM) [18] appliquent la même idée de factorisation pour représenter une distribution de probabilités d'un ensemble de variables aléatoires. Ils s'expriment à l'aide de deux principaux cadres : celui des réseaux Bayésiens (BN) et celui des champs aléatoires de Markov (MRF). Le problème de trouver une affectation des variables de probabilité maximum, appelé *Maximum Probability Explanation* pour BN ou *Maximum A Posteriori* pour MRF, a de nombreuses applications en particulier en traitement d'images et en bioinformatique. Une simple transformation $(-\log)$ convertit ces problèmes en WCSP.

Les modèles graphiques peuvent aussi être modélisés en programmation linéaire à variables 0/1 (01LP), communément utilisée en recherche opérationnelle (RO). Nous considérons deux encodages par la suite. L'un deux se réfère à la notion de *polytope local* [14, 18]. Les algorithmes de propagation de messages, utilisés pour les PGM, peuvent s'interpréter comme des algorithmes de descente de gradient par bloc dans le dual de ce polytope [14]. Ce constat s'applique aussi aux cohérences locales souples dans les modèles graphiques déterministes [8].

Pour les expérimentations, nous avons collecté des instances de modèles graphiques déterministes et probabilistes issues de différentes sources incluant des compétitions CSP, Max-SAT, PPC et PGM. Ces instances ont été encodées dans plusieurs langages de l'IA (WCSP, Max-SAT, MRF), en PPC et en RO (01LP). Habituellement, ces compétitions se restreignent à une famille de méthodes d'optimisation associée à un langage particulier. Au contraire, nous comparons l'efficacité de plusieurs *méthodes complètes* [1] d'optimisation de l'état de l'art pour chacun de ces langages.

2 Langages d'optimisation combinatoire

Dans cette section, nous décrivons les différents langages d'optimisation combinatoire utilisés ensuite. Nous commençons par les modèles graphiques probabilistes qui sont moins connus de la communauté PPC. Nous ne présentons que les champs de Markov car ils n'imposent pas de restriction supplémentaire sur les fonctions locales factorisant la distribution, les réseaux Bayésiens imposant l'utilisation de probabilités conditionnelles avec une exigence de normalisation.

Champ aléatoire de Markov

définition 1 *Un champ aléatoire de Markov (MRF) est défini par une paire (X, Ψ) avec $X = \{x_1, \ldots, x_n\}$, un ensemble de n variables aléatoires et Ψ, un ensemble de* fonctions de potentiel *multiplicatives. Chaque variable $x_i \in X$ a un domaine fini D_i de valeurs qui peuvent lui être affectée. Une fonction de potentiel $\psi_S \in \Psi$, portant sur les variables $S \subseteq X$, est une fonction $\psi_S : D_S \mapsto \mathbb{R}^+$, où D_S exprime le produit Cartésien des domaines D_i pour $x_i \in S$.*

Un champ aléatoire de Markov discret définit une distribution non-normalisée sur X. Pour passer à une distribution de probabilité, on note $P(t)$ la probabilité d'un n-uplet (tuple) donné $t \in D_X$ comme étant :

$$P(t) = \frac{\prod_{\psi_S \in \Psi} \psi_S(t[S])}{Z} = \frac{\exp(-\sum_{\phi_S \in \Phi} \phi_S(t[S]))}{Z}$$

avec $Z = \sum_{t \in D_X} \prod_{\psi_S \in \Psi} \psi_S(t[S])$, une constante de normalisation et $t[S]$, la projection d'un tuple t sur l'ensemble des variables S. La fonction $\phi_S = -\log(\psi_S)$ est appelée fonction de potentiel additive ou aussi *énergie*, en relation avec la physique statistique.

Dans cet article, nous ne considérons que le problème d'optimisation MAP (maximum a posteriori) qui consiste à trouver une affectation complète de probabilité maximum (équivalent à une solution d'énergie minimum). Le problème MAP peut être résolu par des logiciels comme `daoopt` [26], gagnant du PASCAL Probabilistic Inference Challenge (PIC) [2] en 2011, ou `mplp2` [32, 31] qui exploite un algorithme de descente de gradient par bloc pour approcher la résolution d'un système linéaire associé au problème MRF.

exemple 1 *Soit le problème MRF avec deux variables $\{x, y\}$, $D_x = \{a, b\}, D_y = \{a, b, c\}$ et une fonction de potentiel multiplicative $\psi(x, y)$, avec $\psi(a, a) = \psi(b, b) = 1$, $\psi(a, b) = \psi(b, a) = 0.5$, $\psi(a, c) = \psi(b, c) = 0$. L'affectation $(x = a, y = a)$ a une probabilité normalisée maximum $\frac{1}{3}$.*

Problème de satisfaction de contraintes pondérées

Le problème de satisfaction de contraintes pondérées (Weighted CSP) étend le formalisme CSP en remplaçant les contraintes par des fonctions de coût à valeurs entières positives.

définition 2 *Un problème de satisfaction de contraintes pondérées (WCSP) est défini par un triplet (X, W, k) avec $X = \{x_1, \ldots, x_n\}$, un ensemble de n variables discrètes, W, un ensemble de fonctions de coût positives et k, un coût maximum éventuellement infini. Chaque variable $x_i \in X$ a un domaine fini D_i de valeurs qui peuvent lui être affectée. Une fonction $w_S \in W$, portant sur les variables $S \subseteq X$, est une fonction $w_S : D_S \mapsto \{\alpha \in \mathbb{N} \cup \{k\} \ : \ \alpha \leq k\}$.*

Le paramètre k est associé aux tuples interdits permettant de représenter des contraintes. Dans les WCSP, la somme des coûts est bornée par $k : \alpha +_k \beta = \min(\alpha + \beta, k)$. Le coût d'une affectation complète, à minimiser, est égal à la somme bornée de toutes les fonctions de coût. Les WCSP peuvent être résolus par des logiciels comme `toulbar2` [22, 13, 11], gagnant des compétitions Max-CSP [3] en 2008 et MRF en 2010 (UAI 2010 Evaluation [4]).

1. Les méthodes ne sont pas comparées sur la qualité des solutions trouvées mais sur leur capacité à trouver des solutions optimales et prouver leur optimalité dans un temps limité.

2. `http://www.cs.huji.ac.il/project/PASCAL/realBoard.php`

3. `http://www.cril.univ-artois.fr/CPAI08/`

4. `http://www.cs.huji.ac.il/project/UAI10`

Max-SAT Lorsque l'on se restreint à des domaines Booléens avec un langage de clauses pondérées, le problème WCSP devient un problème Max-SAT.

définition 3 *Un problème Max-SAT (Weighted Partial Max-SAT) est défini par un ensemble de paires* $\langle C, w\rangle$*, avec* C*, une clause et* $w \in \mathbb{N} \cup \{\infty\}$*, un nombre appelé le* poids *de la clause. Une clause est une disjonction de littéraux. Un littéral est une variable Booléenne ou sa négation.*

Si le poids d'une clause est infini alors la clause est dite *dure*, sinon elle est *souple*. L'objectif est de trouver une affectation de toutes les variables apparaissant dans les clauses telle que toutes les clauses dures soient satisfaites et que le poids total des clauses souples insatisfaites soit minimum. Parmi les logiciels dédiés à Max-SAT, notons `maxhs` [9, 10], gagnant de la compétition Max-SAT [5] en 2013 en catégorie *crafted WPMS*.

Programmation linéaire en nombres entiers Un problème de programmation linéaire à variables 0/1 (01LP) est défini par un critère linéaire et un système d'équations et inéquations linéaires sur un ensemble de variables Booléennes. Le but est de minimiser le critère tout en satisfaisant l'ensemble des contraintes. Une référence parmi les solveurs 01LP est le logiciel `cplex` d'IBM-ILOG.

Programmation par contraintes Un problème de programmation par contraintes (PPC) est défini par un ensemble de variables discrètes et un ensemble de contraintes. Le but est de minimiser une variable objectif donnée tout en satisfaisant les contraintes. Notons les solveurs PPC `numberjack` `mistral`, `gecode` et `opturion/cpx`, gagnants respectivement des compétitions CSP 2009 [6], MiniZinc 2012 et MiniZinc 2013 [7].

3 Traduction entre les divers formalismes

Dans cette section, nous présentons l'encodage des différents modèles graphiques issus de l'IA/RO/PPC permettant de passer d'un langage à un autre. Pour les conversions, nous avons appliqué une stratégie en étoile en utilisant le formalisme WCSP comme encodage central.

5. `http://maxsat.ia.udl.cat:81/13/benchmarks/`
6. `http://www.cril.univ-artois.fr/CPAI09`
7. `http://www.minizinc.org/challenge201X/results201X.html` with $X \in \{2, 3\}$.

Champ aléatoire de Markov A partir des définitions d'un problème MRF et WCSP, il est clair que la seule différence porte sur le domaine des fonctions : MRF utilise des fonctions potentielles à valeurs réelles tandis que WCSP est habituellement restreint à l'utilisation d'entiers positifs [8].

La conversion d'une instance WCSP en une instance MRF s'effectue par un passage à l'exponentiel des coûts. La base de l'exponentiel est choisie de manière à avoir le potentiel multiplicatif le plus grand égal à 1. Le coût interdit k correspond à un potentiel multiplicatif à 0, préservant ainsi l'effacement de valeurs dans les domaines des variables. Le résultat de la conversion est une instance MRF dans le format UAI "MARKOV" [9].

A l'inverse, pour passer d'une instance MRF vers une instance WCSP, nous employons une représentation des énergies à virgule fixe (précision de chaque énergie à 2 chiffres après la virgule dans les expérimentations). Un décalage constant sur les valeurs résultantes est effectué pour chaque fonction de coût de manière à n'avoir que des entiers positifs.

exemple 2 *La fonction potentielle de l'exemple 1 se traduit en une fonction de coût* $f(a,a) = f(b,b) = 0$*,* $f(a,b) = f(b,a) = -100\log(0.5) = 30$*,* $f(a,c) = f(b,c) = +\infty$*. L'affectation* $(x = a, y = a)$ *a un coût minimum* 0*.*

Max-SAT Une instance Max-SAT est un cas particulier d'une instance WCSP (les poids des clauses étant aussi des entiers positifs). A l'inverse, nous avons repris deux encodages existants de CSP vers SAT : l'encodage *direct* [3] et l'encodage de *tuple* [4].

Encodage direct : pour chaque variable x_i ayant un domaine de taille $|D_i| > 2$, nous avons une proposition d_{ir} pour chaque valeur $r \in D_i$. Cette proposition est vraie ssi la variable x_i est affectée à la valeur r. Pour cela, nous avons les clauses dures $(\neg d_{ir} \vee \neg d_{is})$, $\forall x_i \in \{x_1, \ldots, x_n\}$, $\forall r < s$, $r, s \in D_i$ (clauses *At Most One*), ainsi qu'une clause dure $(\bigvee_r d_{ir})$, $\forall x_i$ (clauses *At Least One*). Ces clauses imposent qu'exactement une seule valeur soit choisie par variable du WCSP. Les variables Booléennes sont directement encodées par une seule proposition sans clause AMO/ALO. Enfin, nous ajoutons une clause $(\bigvee_{x_i \in S} \neg d_{it[x_i]})$ de poids $w_S(t)$, $\forall w_S \in W$, $\forall t \in D_S$ avec $w_S(t) > 0$.

exemple 3 *L'exemple 1 se traduit par quatre propositions* x, y_a, y_b, y_c*, les clauses AMO/ALO* $(\neg y_a \vee \neg y_b)$*,* $(\neg y_a \vee \neg y_c)$*,* $(\neg y_b \vee \neg y_c)$*,* $(y_a \vee y_b \vee y_c)$ *et les clauses pondérées* $(x \vee \neg y_b, 30), (x \vee \neg y_c, \infty), (\neg x \vee \neg y_a, 30), (\neg x \vee \neg y_c, \infty)$*.*

8. A noter que des rationnels positifs sont aussi utilisés dans [8].
9. `http://graphmod.ics.uci.edu/uai08/FileFormat`

Encodage de tuple : le même encodage des domaines que précédemment est effectué, ainsi que pour les fonctions de coût d'arité nulle ou portant sur une seule variable. Nous introduisons une proposition $p_{S,t}$, $\forall w_S \in W$, $\forall t \in D_S$ uniquement lorsque $|S| > 1$ et $w_S(t) < k$. Si $w_S(t) > 0$, nous ajoutons la clause $(\neg p_{S,t})$ avec le poids $w_S(t)$. Cela représente le coût à payer si le tuple t est utilisé. De plus, nous avons une clause dure $(\neg p_{S,t} \vee d_{it[x_i]})$, $\forall x_i \in S$. Si le tuple t est utilisé, alors les valeurs correspondantes $t[x_i]$ doivent être affectées. Enfin, nous avons une clause dure $(\neg d_{ir} \vee \bigvee_{t \in D_S, t[x_i]=r, w_S(t)<k} p_{S,t})$, $\forall w_S \in W, |S| > 1$, $\forall x_i \in S$, $\forall r \in D_i$. Ces clauses imposent que si une valeur $r \in D_i$ est affectée à x_i, un des tuples $t \in D_S$ avec $t[x_i] = r$ doit être utilisé (ou s'ils sont tous interdits, cette valeur ne peut être affectée). Cet encodage a été proposé initialement pour encoder un CSP en SAT [4]. Il permet que la propagation unitaire sur l'encodage de tuple *supprime* les mêmes valeurs que la cohérence d'arc appliquée au CSP d'origine.

exemple 4 *L'exemple 1 se traduit par huit propositions* $x, y_a, y_b, y_c, p_{x_a y_a}, p_{x_a y_b}, p_{x_b y_a}, p_{x_b y_b}$, *les clauses AMO/ALO* $(\neg y_a \vee \neg y_b)$, $(\neg y_a \vee \neg y_c)$, $(\neg y_b \vee \neg y_c)$, $(y_a \vee y_b \vee y_c)$, *les clauses pondérées* $(\neg p_{x_a y_b}, 30)$, $(\neg p_{x_b y_a}, 30)$ *et les clauses dures* $(\neg p_{x_a y_a} \vee \neg x)$, $(\neg p_{x_a y_a} \vee y_a)$, $(\neg p_{x_a y_b} \vee \neg x)$, $(\neg p_{x_a y_b} \vee y_b)$, $(\neg p_{x_b y_a} \vee x)$, $(\neg p_{x_b y_a} \vee y_a)$, $(\neg p_{x_b y_b} \vee x)$, $(\neg p_{x_b y_b} \vee y_b)$ *et* $(x \vee p_{x_a y_a} \vee p_{x_a y_b})$, $(\neg x \vee p_{x_b y_a} \vee p_{x_b y_b})$, $(\neg y_a \vee p_{x_a y_a} \vee p_{x_b y_a})$, $(\neg y_b \vee p_{x_a y_b} \vee p_{x_b y_b})$, $(\neg y_c)$.

Le résultat de ces deux encodages est donné dans le format `wcnf` [10], qui inclut l'information du majorant k, de manière à préserver la propagation.

La complexité asymptotique des deux encodages est la même en terme de nombre de tuples à représenter explicitement : $O(nd^2 + ert)$ avec n, le nombre de variables, d, la taille du plus grand domaine, e, le nombre de fonctions de coût, r, l'arité maximum des fonctions de coût, et t, le nombre maximum de tuples par fonction, avec l'hypothèse que $t > d$. Cependant cette notation asymptotique cache une constante plus grande pour l'encodage de tuple. Or, le nombre de tuples devant être explicitement représentés dans l'encodage direct est le nombre de tuples ayant un coût non nul, alors que c'est le nombre de tuples ayant un coût inférieur à k dans l'encodage de tuple. Dans nos expérimentations, nous observons que l'encodage de tuple génère des fichiers bien plus volumineux que l'encodage direct, indiquant que pour la plupart des instances, il y a beaucoup plus de tuples de coût zéro que de coût infini (k).

10. `http://www.maxsat.udl.cat/08/index.php?disp=requirements`

Programmation linéaire en nombres entiers L'encodage d'une instance WCSP en 01LP [11] est similaire à celui fait pour Max-SAT. Des variables 0/1 remplacent les variables propositionnelles. Cependant des différences mineures apparaissent dues au pouvoir d'expression supérieur des contraintes linéaires par rapport aux clauses.

Encodage direct : les clauses AMO/ALO sont remplacées par une seule contrainte linéaire : $\sum_{r \in D_i} d_{ir} = 1, \forall x_i \in X$ avec $|D_i| > 2$. La fonction objectif d'un WCSP n'étant pas forcément linéaire, cela nécessite d'introduire des variables 0/1 $p_{S,t}$, $\forall w_S \in W$, $\forall t \in D_S$ avec $0 < w_S(t) < k$. La fonction objectif de la formulation 01LP est alors $\sum w_S(t) p_{S,t}$. Nous ajoutons une contrainte linéaire $\sum_{x_i \in S}(1 - d_{it[x_i]}) + p_{S,t} \geq 1$ qui impose que la variable $p_{S,t} = 1$ si le tuple t est utilisé. Si $w_S(t) = k$, alors la contrainte devient $\sum_{x_i \in S}(1 - d_{it[x_i]}) \geq 1$ et le terme $w_S(t) p_{S,t}$ correspondant disparaît de l'objectif.

exemple 5 *L'exemple 1 se traduit par six variables 0/1* $x, y_a, y_b, y_c, p_{x_a y_b}, p_{x_b y_a}$, *les contraintes* $y_a + y_b + y_c = 1$, $x - y_b + p_{x_a y_b} \geq 0$, $-x - y_a + p_{x_b y_a} \geq -1$, $x - y_c \geq 0$, $-x - y_c \geq -1$ *et la fonction objectif* $30 p_{x_a y_b} + 30 p_{x_b y_a}$.

Encodage de tuple : nous utilisons le même encodage que précédemment pour exprimer les domaines et les fonctions de coût d'arité 0/1. Nous ajoutons une contraint linéaire $d_{ir} = \sum_{t \in D_S, t[x_i]=r, w_S(t)<k} p_{S,t}$, $\forall w_S \in W, |S| > 1$, $\forall x_i \in S, \forall r \in D_i$ qui impose que $d_{ir} = 1$ ssi il existe un tuple t tel que $t[x_i] = r$ et $w_s(t) < k$ (et $d_{ir} = 0$ si tous les tuples sont interdits). La fonction objectif est identique à l'encodage direct.

exemple 6 *L'exemple 1 se traduit par huit variables 0/1* $x, y_a, y_b, y_c, p_{x_a y_a}, p_{x_a y_b}, p_{x_b y_a}, p_{x_b y_b}$, *les contraintes* $y_a + y_b + y_c = 1$, $1 - x = p_{x_a y_a} + p_{x_a y_b}$, $x = p_{x_b y_a} + p_{x_b y_b}$, $y_a = p_{x_a y_a} + p_{x_b y_a}$, $y_b = p_{x_a y_b} + p_{x_b y_b}$, $y_c = 0$ *et la fonction objectif* $30 p_{x_a y_b} + 30 p_{x_b y_a}$.

Cet encodage de tuple a été proposé par Koster [20]. Il est équivalent à l'encodage en 01LP couramment utilisé pour les PGM dans le cas de fonctions d'arité au plus 2 [18] (chapitre 13 section 5). Il est facile de voir que la contrainte d'intégralité sur les variables $p_{S,t}$ peut être relâchée dans les deux encodages : si toutes les variables d_{ir} sont affectées à 0 ou 1, alors les contraintes et la fonction objectif assurent que les $p_{S,t}$ sont fixées à 0 ou 1. Dans le format `cplex` "LP" généré, nous relaxons cette contrainte d'intégralité.

Il est intéressant de noter que la relaxation continue de l'encodage de tuple est équivalente au *polytope local* décrit pour les MRFs [33, 14] et précédemment

11. Nous n'étudions pas l'encodage inverse 01LP vers WCSP.

étudié par Schlesinger en traitement d'images pour le cas de grammaires de motifs 2D [30]. C'est aussi équivalent au dual du programme linéaire lié à la cohérence d'arc optimale souple (OSAC) [7, 8]. Le minorant produit par OSAC est supérieur à ceux produits par les autres cohérences d'arc souples comme EDAC [22], à l'exception des possibles interactions avec la cohérence de noeud. Ce minorant dual a récemment été montré comme étant très expressif au sens où n'importe quel programme linéaire peut être traduit en temps linéaire dans le problème de calculer ce minorant pour un modèle graphique adapté [19].

Programmation par contraintes Dans [27], un encodage d'une instance WCSP en PPC a été proposée. Les variables de décision sont identiques au WCSP. Chaque fonction de coût est réifiée en une contrainte qui s'applique aux variables de la fonction et à une variable supplémentaire représentant le coût de l'affectation. Le problème résultant est un CSP avec plus de variables et des arités accrues. Typiquement, les fonctions de coût d'arité 1 et 2 sont traduites en des contraintes de `table` d'arité 2 et 3 respectivement. Enfin une variable objectif supplémentaire est reliée par une contrainte de `somme` à toutes les autres variables de coût. Toutes ces variables supplémentaires ont un domaine de valeurs entières positives borné par le majorant initial k. La même approche est employée pour encoder une instance Max-SAT, à l'exception des contraintes de `table` qui sont remplacées par des expressions Booléennes réifiées encodant les clauses dures et souples. Le résultat est exprimé dans le langage PPC `minizinc` [25].

A l'inverse, traduire une instance PPC en WCSP est une tâche plus difficile. Il s'agit de retrouver la factorisation de la fonction objectif en une somme de fonctions de coût locales, à partir de la variable objectif, tout en éliminant les variables intermédiaires. Pour cela, nous avons développé un prototype dans `numberjack` [15] lisant le format bas-niveau `flatzinc` [12] [25]. De plus, les contraintes globales ont été décomposées en fonctions de coût ($\{0, \infty\}$) d'arité 3 [1].

4 Comparaison de logiciels d'optimisation

4.1 Collection de benchmarks

Nous avons collecté 3018 instances provenant de différentes sources de modèles graphiques déterministes (WCSP, CSP, Max-SAT), probabilistes (MRF) et aussi de compétitions PPC. Elles sont disponibles aux formats `uai`, `wcsp`, `wcnf`, `lp` et `minizinc` [13].

CFLib [14] est une collection de problèmes WCSP. Nous en avons extrait une partie, codée nativement au format `wcsp` et recodée manuellement au format `minizinc`. Cela inclut les enchères combinatoires [23], l'allocation de fréquence CELAR/GRAPH [6], la correction d'erreur Mendélienne [29], la conception de protéines [2], la sélection de prises de vue du satellite SPOT5 [5] et l'allocation d'entrepôts [21, 22].

Les instances probabilistes MRF proviennent des compétitions UAI 2008 (problème d'analyse de liaison génétique [15] [13], arité 5) et PIC 2011 (arité 2), au format natif `uai` (converties à 2 chiffres après la virgule). D'autres instances d'arité 2 et 3 sont issues d'un benchmark OpenGM2 Computer Vision and Pattern Recognition (CVPR) [16] [16] au format natif `HDF5` contenant des énergies au lieu des probabilités. ColorSeg, MatchingStereo, PhotoMontage ont des énergies entières directement traduites en fonctions de coût, les autres problèmes étant convertis à 8 chiffres après la virgule (instances limitées à 1Go).

Les instances Max-SAT proviennent des catégories Crafted (W)PMS (MIPLib & DIMACS Max Clique) et Industrial WPMS [17]. La conversion au format `wcsp` permet d'encoder chaque clause par une fonction de coût avec un seul tuple de coût non nul. Les traductions vers la PPC (resp. MRF) imposent que les coûts tiennent sur 32 bits (resp. les fonctions de coût soient de faible arité).

Nous avons sélectionné des problèmes CSP binaires définis en extension (*i.e.,* sans contrainte globale) et comportant des instances insatisfiables (BlackHole, Langford, Quasi-group Completion Problem, coloriage de graphe et problèmes aléatoires Composed, 3-SAT EHI et Geometric) des compétitions CSP & Max-CSP [18]. Ces instances au format natif `xcsp2.1/xml` ont été transformées en WCSP (Max-CSP) tels que chaque tuple permis (resp. interdit) d'une contrainte a un coût nul (resp. unitaire) dans la fonction de coût correspondante. Nous fixons $k = 1000$. Enfin, nous avons pris des problèmes PPC décomposables en WCSP des compétitions MiniZinc 2012 & 2013.

Dans la Table 1, nous indiquons les tailles maximum des instances par problème. Dans le cas des problèmes issus de la PPC, ces tailles correspondent au sous-ensemble des instances décomposables (seule une partie indiquée entre parenthèses des instances FastFood, Golomb et OnCallRostering a pu être convertie au format `wcsp` en utilisant moins de 1 Go par instance).

12. Dans les expérimentations, une limite de 20 minutes a été imposée lors de la traduction de `minizinc` à `flatzinc`.
13. `http://genoweb.toulouse.inra.fr/~degivry/evalgm`
14. `http://costfunction.org/benchmark`
15. `http://graphmod.ics.uci.edu/uai08/Evaluation/Report/Benchmarks`
16. `http://hci.iwr.uni-heidelberg.de/opengm2`
17. `http://maxsat.ia.udl.cat:81/13/benchmarks/`
18. `http://www.cril.univ-artois.fr/CPAI08/, ../~lecoutre/benchmarks.html`

Nous donnons également les majorants initiaux maximum k, arbitrairement fixés à une grande valeur ou à la somme du maximum de chaque fonction de coût plus un, ainsi qu'une sur-approximation de l'ensemble des coûts utilisés dans un problème avec ∞ indiquant l'occurrence de tuples interdits (contraintes).

Les plus grandes instances en nombre de variables proviennent des benchmarks Max-SAT et CVPR (presque 1 million de variables pour Max-SAT/TimeTabling et un demi-million pour CVPR/PhotoMontage et ColorSeg). De plus, Max-SAT contient des clauses portant sur un grand nombre de variables (580 dans Haplotyping). Pour les autres benchmarks, l'arité des fonctions reste faible entre 2 et 5. La connectivité du graphe des modèles graphiques probabilistes (MRF/CVPR) et Max-SAT est souvent très faible (utilisation d'une structure de grille pour CVPR). Cependant MRF/ObjectDetection, WCSP/ProteinDesign, Max-CSP/Langford et CVPR/Matching ont un graphe complet. MRF/ProteinFolding a le plus grand domaine (503 valeurs). La plupart des instances CVPR utilisent des coûts dans un intervalle très large (du fait de la précision à 8 chiffres), tandis que les instances Max-CSP ne contiennent que des coûts 0/1. Tous les modèles graphiques déterministes, à l'exception de Max-CSP et WCSP/CELAR, ont des tuples interdits. A l'inverse, les modèles probabilistes MRF & CVPR n'en ont pas (exceptés MRF/Linkage & DBN).

4.2 Logiciels et paramètres expérimentaux

Nous avons comparé plusieurs logiciels de l'état de l'art en optimisation combinatoire : `daoopt` [19] (réglage des paramètres à 20 min. sauf pour CVPR à 1 heure [26], sans amélioration des bornes par MPLP), `toulbar2` [20] (paramètres `-l=1 -dee=1`), `mplp2` [21] (seuil de précision interne à 2.10^{-7}), `maxhs` (pas de paramètre), `numberjack mistral` [22], `gecode` [23], `opturion/cpx` [24] (mode *free search*) et `cplex` 12.4 (paramètres EPAGAP, EPGAP, and EPINT mis à zéro pour éviter un arrêt intempestif).

Tous les calculs ont été réalisés sur un seul coeur AMD Operon 6176 à 2.3 GHz et 8 Go de mémoire.

4.3 Résultats expérimentaux

En Table 2 figure le nombre d'instances résolues en moins de 20 min. [25] (excepté 1 heure pour CVPR) [26].

Bien que le meilleur solveur en nombre d'instances résolues par classe de problèmes appartienne à cette classe, *i.e.,* `toulbar2` pour WCSP, `maxhs` pour Max-SAT, `gecode` pour PPC et `mplp2` pour CVPR, les résultats montrent que certains solveurs comme `maxhs`, `toulbar2` et `cplex` marchent bien sur plusieurs classes, résolvant resp. 1942, 2191 et 2323 instances parmi 3018, gagnant la première place pour 12, 17, 18 problèmes respectivement [27] parmi 46 catégories de problèmes. CVPR/ColorSeg est le problème avec l'espace de recherche le plus grand ($d^n = 2^{829440}$) complètement résolu par `mplp2`. MRF/ObjectDetection est le problème avec l'espace de recherche le plus petit entièrement non résolu ($d^n \approx 2^{263}$).

`toulbar2` a obtenu la première place pour quatre problèmes MRF (hors CVPR). L'analyse des temps de calcul (figures non disponibles) sur le problème Segmentation montre que `toulbar2` est souvent le plus rapide mais il est dominé par `mplp2` et `cplex`$_t$ (avec l'encodage de tuple) sur les instances les plus difficiles. Ici, `cplex` avec l'encodage direct et `maxhs` ne résolvent aucune instances avec une taille de domaine $d = 21$ et `daoopt` seulement 7. Sur Linkage [17], `cplex`, suivi par `maxhs`, obtient de très bons résultats, montrant sa capacité à traiter des fonctions de coût d'arité supérieure (5). A noter qu'un solveur Max-SAT, `maxhs` [9, 10], utilisant des techniques peu connues de la communauté MRF, obtient de bons résultats sur les modèles graphiques probabilistes (au moins pour une précision à 2 chiffres). De manière surprenante, l'encodage direct donne de meilleurs résultats sur le problème Grid ($d = 2$) pour `cplex` qui exploite un grand nombre de coupes *zero-half*. Les solveurs PPC obtiennent de mauvais résultats sur MRF en partie du fait de l'absence de contraintes dures (excepté pour Linkage) et de la faiblesse des minorants obtenus par propagation sur les variables de coût, ce phénomène étant aussi observé pour le cas des fonctions de coût globales dans [24].

Sur le benchmark Max-SAT, les clauses d'arité large interdisent d'appliquer l'encodage de tuple (trop grand nombre de tuples de coût nul) ni d'exprimer les problèmes en utilisant des tables en extension comme pour le format `uai`. Il est intéressant de noter la complémentarité de `cplex` et `maxhs` (qui utilise aussi `cplex` pour extraire un minorant à partir de noyaux insatisfiables identifiés par `minisat`) suivant les problèmes. Sur le problème Upgradeability, `cplex` (resp. `toulbar2`) est jusqu'à deux (resp. un) ordre(s) de grandeur plus rapide que `maxhs`. Dans PlanningWPref, `opturion/cpx`, un solveur PPC intégrant un mécanisme d'apprentissage à partir des conflits, obtient la seconde place derrière `maxhs`, résolvant toutes les instances (ex-

19. `https://github.com/lotten/daoopt` version 1.1.2.
20. `mulcyber.toulouse.inra.fr/projects/toulbar2` v0.9.6
21. `http://cs.nyu.edu/~dsontag/` version 2.
22. `http://numberjack.ucc.ie/` version 1.3.40.
23. `http://www.gecode.org/` version 4.2.0.
24. `http://www.opturion.com` version 1.0.2.
25. Sans les temps de conversion entre formats.
26. Des résultats détaillés par instance et des figures *cactus plot* à `http://genoweb.toulouse.inra.fr/~degivry/evalgm`.
27. En prenant le meilleur encodage à chaque fois pour `maxhs` et `cplex`.

cepté WCNF_storage_p03) qui ont pu être traduites dans le format bas-niveau `flatzinc` en moins de 20 minutes, incluant une instance avec 10946 variables (WCNF_pathways_p18).

Concernant le benchmark WCSP, `toulbar2` domine clairement sur les problèmes CELAR, Pedigree et ProteinDesign, tandis que `cplex` avec l'encodage direct, suivi par `maxhs`, l'emporte sur les problèmes Auction et Warehouse issus de la Recherche Opérationnelle. L'encodage de tuple en 01LP se comporte favorablement si la taille des instances est relativement faible ($n \times d \leq 20,000$), sinon des problèmes d'allocation mémoire apparaissent comme c'est le cas pour les plus grandes instances Warehouse.

Les bonnes performances de `maxhs` dans le benchmark Max-CSP peuvent s'expliquer par sa capacité à bien résoudre toutes les instances satisfiables (optimum à zéro) en particulier pour les problèmes Geometric et QCP grâce à son solveur SAT interne `minisat`. Les bons résultats obtenus par `daoopt` peuvent également s'expliquer par sa phase initiale de recherche locale stochastique [26], trouvant de bons majorants pour la phase suivante de recherche arborescente, spécialement sur les instances aléatoires comme Geometric et EHI. `toulbar2` et `cplex` gagnent la première place sur quatre problèmes Max-CSP, obtenant des performances comparables sur Coloring. Pour le problème Composed, `toulbar2` domine souvent en temps de plusieurs ordres de grandeur par rapport aux autres approches à l'exception du solveur PPC `mistral`. Tous les deux utilisent une heuristique de choix de variable *dom/wdeg* exploitant les conflits particulièrement efficace pour ce problème. Enfin, l'encodage de tuple s'avère toujours contre-productif pour ces problèmes.

Les instances PPC sont pour la plupart difficiles à traduire en fonctions de coût locales et avec de petits domaines ($\frac{1}{4}$ des instances n'ont pu être converties à cause de problèmes d'espace mémoire). De plus, le résultat de la conversion n'est souvent pas approprié pour les solveurs 01LP (les contraintes linéaires étant décomposées), ce qui explique les mauvais résultats pour `cplex`. Sur ce benchmark, `gecode` obtient en moyenne les meilleurs résultats. Cependant, un solveur Max-SAT, `maxhs`, le domine sur deux problèmes : Amaze et OnCallRostering. Et `daoopt` est plus rapide que `gecode` sur les instances difficiles de ParityLearning. En effet, `daoopt` résout toutes les instances en prétraitement grâce à l'élimination de variables [12], nécessitant ici un espace mémoire (2Go) juste inférieur à sa limite fixée par sa borne $i = 25$ [26].

Pour comparer les résultats sur le benchmark MRF/CVPR, au lieu de donner les résultats des solveurs PPC (probablement nuls à cause de problèmes de conversion des coûts en simples domaines bornés à 32 bits), nous indiquons les meilleurs résultats obtenus lors d'une évaluation de méthodes développées en traitement d'images [16] [28]. Les résultats montrent une relative bonne performance des solveurs généralistes tels que `mplp2`, `toulbar2` et `cplex` comparés à des solveurs spécialisés en traitement d'images. Sur le problème Scene Decomposition, qui utilise un modèle de *superpixel* [16], `toulbar2` résout toutes les 715 instances en $0,023$ secondes en moyenne comparé à $0,134$ (resp. $1,039$) secondes pour `mplp2` (resp. $\texttt{cplex}_t$). Bien que l'encodage de tuple fut toujours contre-productif pour un solveur Max-SAT sur les autres benchmarks (voir la colonne $\texttt{maxhs}_t$ dans la Table 2), ce n'est pas le cas sur ce problème (et aussi pour GeomSurf-3). La plus grande instance résolue par `toulbar2` est InPainting-4/triplepoint4-plain-ring avec 14400 variables en $1,47$ secondes comparé à $157,22$ (resp. $193,72$) secondes pour `mplp2` (resp. $\texttt{cplex}_t$). Ici les algorithmes spécialisés dominent avec $0,58$ sec. pour la méthode BUNDLE-H, $2,19$ sec. pour TRWS et $12,39$ sec. pour MCA.

5 Conclusion, vers une approche portfolio

En résumé, `toulbar2` obtient les meilleurs résultats lorsque les domaines sont larges ($d > 20$, excepté BlackHole), `maxhs` lorsque l'arité des clauses est grande ($r > 300$), `daoopt` lorsque la structure du modèle graphique est bien décomposable (faible largeur induite), `mplp2` lorsque la relaxation linéaire est proche de l'optimum, `gecode` lorsque le problème est directement modélisé en contraintes. Pour `cplex`, l'encodage de tuple domine l'encodage direct, sauf pour des domaines de taille 2 (ou 3 pour SPOT5) ou des problèmes de taille trop importante ($n \times d \geq 20,000$). C'est l'inverse pour `maxhs`, sauf en traitement d'images.

Les résultats montrant que le meilleur solveur varie beaucoup en fonction de chaque problème, cela suggère d'élaborer une stratégie portfolio. Nous avons utilisé pour cela Numberjack [15] qui offre une interface vers des solveurs PPC, WCSP, LP et SAT. Nous avons intégré la lecture du format `flatzinc` et combiné deux solveurs (`mistral` avec trois politiques de restart différentes et `toulbar2`), exécutés chacun isolément (1 coeur et 8Go) en parallèle, à l'instar de `ppfolio` [28]. Notre approche résout 644 instances hormis MRF/CVPR (`nj-port` en Table 2). Davantage de travail reste à faire pour mieux parser le format `flatzinc` et ajouter d'autres solveurs comme `cplex` avec les deux encodages direct et de tuple.

28. Dans leur expérimentations, ils ont utilisé un processeur Xeon W3550 à 3GHz, 12Go de mémoire vive et 1 heure de temps de calcul.

Problème	Nb.	n	d	e	r	intervalle des coûts utilisés (k)
MRF (`uai`)	319					
Linkage	22	1289	7	2184	5	18 ... ∞ (1000000)
DBN	108	1094	2	22793	2	7 ... ∞ (100000000)
Grid	21	6400	2	19200	2	27 ... 2967 (100000000)
ImageAlignment	10	400	93	3563	2	907 ... 2291 (100000000)
ObjectDetection	37	60	21	1830	2	530 ... 27860 (100000000)
ProteinFolding	21	1972	503	8816	2	53 ... 23327 (100000000)
Segmentation	100	237	2/21	886	2	1 ... 1166 (100000000)
Max-SAT (`wcnf`)	427					
MIPLib	12	24776	2	107956	93	1 ... ∞ (547769)
MaxClique	62	3321	2	378247	2	1 ... ∞ (3321)
Haplotyping	100	216117	2	1188220	580	1 ... ∞ (10691000)
PackupWeighted	99	25554	2	70677	177	60 ... ∞ (27206200)
PlanningWithPref	29	69409	2	771883	372	3000 ... ∞ (100000)
TimeTabling	25	903884	2	2912880	36	2 ... ∞ (14850)
Upgradeability	100	18169	2	105097	77	1 ... ∞ (61594000000)
WCSP (`wcsp`)	281					
Auction	170	246	2	11528	2	36 ... ∞ (249794)
CELAR	16	458	44	2335	2	1 ... 2020000 (468527000)
Pedigree	10	10017	28	18875	3	1 ... ∞ (2484)
ProteinDesign	10	18	198	171	2	2304 ... ∞ (28925100)
SPOT5	20	1057	4	21786	3	2 ... ∞ (635869)
Warehouse	55	1100	300	101100	2	5124 ... ∞ (238093000)
Max-CSP (`xcsp`)	503					
BlackHole	37	205	50	1651	2	1 (1000)
Coloring	22	450	6	6164	2	1 (1000)
Composed	80	83	10	785	2	1 (1000)
EHI	200	315	7	4715	2	1 (1000)
Geometric	100	50	20	605	2	1 (1000)
Langford	4	33	29	517	2	1 (1000)
QCP	60	264	9	2662	2	1 (1000)
PPC (`minizinc`)	35					
AMaze	6	1573	17	3173	4	1 ... ∞ (100000000)
FastFood	6(1)	2	5	3	2	1 ... ∞ (100000000)
Golomb	6(3)	44	163	717	3	1 ... ∞ (100000000)
OnCallRostering	5(3)	2205	89	4513	4	1 ... ∞ (100000000)
ParityLearning	7	759	20	1440	4	1 ... ∞ (100000000)
VehicleRoutingProb.	5	11531	100	22999	4	12 ... ∞ (100000000)
MRF/CVPR (`hdf5`)	1453					
ChineseChars	100	17856	2	553726	2	17432999 ... 723775999 (210467017114895)
ColorSeg	3	414720	4	2069714	2	5 ... 1280 (632538770)
ColorSeg-4	9	86400	12	258600	2	3717 ... 300000000 (25526293828226)
ColorSeg-8	9	86400	12	430202	2	3717 ... 300000000 (24010445405740)
GeomSurf-3	300	1133	3	5039	3	33175 ... 1517620002 (1082559978572)
GeomSurf-7	300	1133	7	5039	3	406409 ... 1517620002 (1703380258076)
InPainting-4	2	14400	4	42960	2	78539816 ... 10000000000 (118363097154663)
InPainting-8	2	14400	4	71282	2	27768018 ... 10000000000 (118027994393422)
Matching	4	20	20	210	2	3 ... 1000000000000000 (190000000000000000)
MatchingStereo	2	166222	20	497849	2	1 ... 765 (149114978)
ObjectSeg	5	68160	8	203947	2	99 ... 620000000 (77323343626999)
PhotoMontage	2	514080	7	1540689	2	1 ... 100000 (152615400000)
SceneDecomp	715	208	8	769	2	187000 ... 3529616658 (1225793288120)

TABLE 1 – Tailles maximum par problème (n, nombre de variables, d, taille des domaines, e, nombre de fonctions de coût, r, arité des fonctions de coût) et intervalles des coûts avec le majorant initial entre parenthèses (k).

Problème	Nb.	daoopt	mplp2	toulbar2	cplex	cplex$_t$	maxhs	maxhs$_t$	opturion	gecode	mistral	nj-port
MRF (`uai`)	319	144	123	**219**	152	205	104	72	1	0	1	181
Linkage	22	16	1	13	14	**22**	20	20	0	0	1	10
DBN	108	60	0	**77**	64	65	30	0	0	0	0	70
Grid	21	2	2	0	**15**	0	4	2	0	0	0	0
ImageAlignment	10	9	**10**	**10**	0	9	0	0	0	0	0	5
ObjectDetection	37	0	0	0	0	0	0	0	0	0	0	0
ProteinFolding	21	0	10	**19**	9	9	0	0	0	0	0	5
Segmentation	100	57	**100**	**100**	50	**100**	50	50	1	0	0	91
Max-SAT (`wcnf`)	427	11	0	195	269	N/A	**277**	N/A	27	19	15	45
MIPLib	12	2	0	**3**	**3**	N/A	**3**	N/A	2	**3**	2	**3**
MaxClique	62	9	0	33	**38**	N/A	36	N/A	10	15	13	24
Haplotyping	100	N/A	N/A	1	18	N/A	**25**	N/A	MZN	MZN	MZN	MZN
PackupWeighted	99	N/A	N/A	52	**99**	N/A	85	N/A	1	0	0	18
PlanningWPref	29	N/A	N/A	6	11	N/A	**27**	N/A	14	1	0	0
TimeTabling	25	N/A	N/A	0	0	N/A	**1**	N/A	MZN	MZN	MZN	MZN
Upgradeability	100	N/A	N/A	**100**	**100**	N/A	**100**	N/A	N/A	N/A	N/A	N/A
WCSP (`wcsp`)	281	188	44	**247**	241	235	227	195	70	130	111	179
Auction	170	158	0	167	**170**	**170**	**170**	166	61	104	108	**170**
CELAR	16	3	0	**12**	0	3	0	0	1	0	1	1
Pedigree	10	4	0	**10**	5	9	5	5	4	0	0	0
ProteinDesign	10	4	7	**9**	0	6	0	0	0	0	0	5
SPOT5	20	6	0	4	**16**	10	5	5	1	0	2	3
Warehouse	55	13	37	45	**50**	37	47	19	3	26	0	0
Max-CSP (`xcsp`)	503	173	0	216	199	50	**238**	150	118	6	96	221
BlackHole	37	10	0	10	**30**	10	10	10	10	0	10	**30**
Coloring	22	16	0	**17**	**17**	15	12	12	8	4	6	15
Composed	80	26	0	**80**	**80**	15	**80**	15	**80**	0	**80**	**80**
EHI	200	0	0	0	0	0	0	0	0	0	0	0
Geometric	100	91	0	**93**	49	0	88	77	9	0	0	82
Langford	4	**2**	0	**2**	**2**	1	**2**	**2**	1	**2**	0	0
QCP	60	28	0	14	21	9	**46**	34	10	0	0	14
PPC (`minizinc`)	35	10	1	13	2	5	16	8	18	**26**	18	18
AMaze	6	0	0	2	0	2	**6**	3	5	4	2	2
FastFood	6	1	1	1	1	1	1	1	**6**	**6**	5	5
Golomb	6	0	0	3	0	0	3	1	4	**6**	5	5
OnCallRoster.	5	2	0	2	1	2	**3**	**3**	2	2	2	2
ParityLearning	7	**7**	0	5	0	0	3	0	1	**7**	4	4
VRP	5	0	0	0	0	0	0	0	0	**1**	0	0
CVPR (`hdf5`)	1453	1272	**1339**	1301	372	1329	311	1008	`TRWS`	`BUN-DLE`	`MCA`	`MCBC`
ChineseChars	100	0	0	0	0	0	0	0	0	N/A	N/A	**56**
ColorSeg	3	1	**3**	0	0	1	0	0	1	0	**3**	N/A
ColorSeg-4	9	0	**8**	0	0	3	0	0	7	7	**8**	N/A
ColorSeg-8	9	0	4	0	0	2	0	0	2	1	**8**	N/A
GeomSurf-3	300	**300**	**300**	**300**	**300**	**300**	236	291	N/A	277	N/A	N/A
GeomSurf-7	300	252	**300**	281	72	**300**	74	2	N/A	180	N/A	N/A
InPainting-4	2	0	**1**	**1**	0	**1**	0	0	**1**	**1**	**1**	N/A
InPainting-8	2	0	**1**	0	0	0	0	0	0	0	**1**	N/A
Matching	4	**4**	**4**	**4**	0	3	0	0	0	0	N/A	N/A
MatchingStereo	2	0	0	0	0	0	0	0	0	**1**	N/A	N/A
ObjectSeg	5	0	3	0	0	4	0	0	**5**	3	**5**	N/A
PhotoMontage	2	0	0	0	0	0	0	0	0	0	N/A	N/A
SceneDecomp	715	**715**	**715**	**715**	0	**715**	1	**715**	712	673	N/A	N/A
Nb. of 1st Pos.	46	5	10	**17**	13	7	11	3	2	6	1	4

TABLE 2 – Nombre d'instances résolues en moins de 20 min. (1 heure pour CVPR). Meilleurs résultats en gras.

Remerciements.

Ce travail a été en partie financé par l'Agence nationale de la Recherche, référence ANR-10-BLA-0214. Nous sommes reconnaissant à la plateforme bioinformatique Genotoul de Toulouse pour l'accès au cluster.

Références

[1] D Allouche, C Bessiere, P Boizumault, S Givry, P Gutierrez, S Loudni, JP Métivier, and T Schiex. Decomposing global cost functions. In *Proc. of AAAI*, 2012.

[2] D Allouche, S Traoré, I André, S Givry, G Katsirelos, S Barbe, and T Schiex. Computational protein design as a cost function network optimization problem. In *Proc. of CP*, pages 840–849, 2012.

[3] J Argelich, A Cabiscol, I Lynce, and F Manyà. Encoding Max-CSP into partial Max-SAT. In *Proc. of ISMVL*, pages 106–111, 2008.

[4] F Bacchus. GAC via unit propagation. In *Proc. of CP*, pages 133–147, 2007.

[5] E Bensana, M Lemaître, and G Verfaillie. Earth observation satellite management. *Constraints*, 4(3) :293–299, 1999.

[6] B Cabon, S de Givry, L Lobjois, T Schiex, and J Warners. Radio link frequency assignment. *Constraints*, 4 :79–89, 1999.

[7] M Cooper, S de Givry, and T Schiex. Optimal soft arc consistency. In *Proc. of IJCAI*, pages 68–73, 2007.

[8] M Cooper, S Givry, M Sanchez, T Schiex, M Zytnicki, and T Werner. Soft arc consistency revisited. *Artif. Intell.*, 174 :449–478, 2010.

[9] J Davies and F Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proc. of CP*, pages 225–239, 2011.

[10] J Davies and F Bacchus. Exploiting the power of MIP solvers in MaxSAT. In *Proc. of SAT*, pages 166–181, 2013.

[11] S de Givry, S Prestwich, and B O'Sullivan. Dead-end elimination for weighted CSP. In *Proc. of CP*, pages 263–272, 2013.

[12] R Dechter. Bucket elimination : A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2) :41–85, 1999.

[13] A Favier, S Givry, A Legarra, and T Schiex. Pairwise decomposition for combinatorial optim. in graphical models. In *Proc. of IJCAI*, 2011.

[14] A Globerson and T Jaakkola. Fixing max-product : Convergent message passing algorithms for MAP LP-relaxations. In *Proc. of NIPS*, pages 553–560, 2007.

[15] E Hebrard, E O'Mahony, and B O'Sullivan. Constraint Programming and Combinatorial Optimisation in Numberjack. In *Proc. of CP-AI-OR*, pages 181–185, 2010.

[16] J Kappes, B Andres, F Hamprecht, C Schnörr, S Nowozin, D Batra, S Kim, B Kausler, J Lellmann, N Komodakis, and C Rother. A comparative study of modern inference techniques for discrete energy minimization problem. In *Proc. of CVPR*, 2013.

[17] A Kishimoto and R Marinescu. Recursive best-first and/or search with overestimation for genetic linkage analysis. In *Proc. of CP Workshop on Constraint Based Methods for Bioinformatics*, 2013.

[18] D Koller and N Friedman. *Probabilistic graphical models : principles and techniques*. The MIT Press, 2009.

[19] V Kolmogorov. The power of linear programming for finite-valued csps : a constructive characterization. In *Automata, Lang., and Program.*, pages 625–636. 2013.

[20] A Koster. *Frequency assignment : Models and Algorithms*. PhD thesis, 1999.

[21] J Kratica, D Tošic, V Filipović, and I Ljubić. Solving the simple plant location problem by genetic alg. *RAIRO*, 35(1) :127–142, 2001.

[22] 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 IJCAI*, pages 84–89, 2005.

[23] J Larrosa, F Heras, and S de Givry. A logical approach to efficient max-sat solving. *Artif. Intell.*, 172(2-3) :204–233, 2008.

[24] J Lee and K Leung. Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction. *JAIR*, 43 :257–292, 2012.

[25] N Nethercote, P Stuckey, R Becket, S Brand, G Duck, and G Tack. MiniZinc : Towards a standard CP modelling language. In *Proc. of CP*, pages 529–543, 2007.

[26] L Otten, A Ihler, K Kask, and R Dechter. Winning the PASCAL 2011 MAP challenge with enhanced AND/OR branch-and-bound. In *NIPS DISCML Workshop*, 2012.

[27] T Petit, JC Régin, and C Bessière. Meta constraints on violations for over constrained problems. In *Proc. of ICTAI*, pages 358–365, 2000.

[28] O Roussel. Description of ppfolio. *SAT Competition 2011*, 2011.

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

[30] M.I. Schlesinger. Syntactic analysis of two-dimensional visual signals in noisy conditions. *Kibernetika*, 4 :113–130, 1976.

[31] D Sontag, D Choe, and Y Li. Efficiently searching for frustrated cycles in MAP inference. In *Proc. of UAI*, pages 795–804, 2012.

[32] D Sontag, T Meltzer, A Globerson, Y Weiss, and T Jaakkola. Tightening LP relaxations for MAP using message-passing. In *Proc. of UAI*, 2008.

[33] T Werner. A linear programming approach to max-sum problem. *Pattern A. & Mach. Int.*, 29(7), 2007.