Comparaison de logiciels d'optimisation discrète sur une collection multi-langage de modèles graphiques

Barry Hurley [1], **Barry O'Sullivan** [1], **David Allouche** [2], **George Katsirelos** [2], **Thomas Schiex** [2], **Matthias Zytnicki** [2], **Simon de Givry** [2]

1. *Insight Centre for Data Analytics, University College Cork, Ireland*
   firstname.lastname@insight-centre.org

2. *MIAT, UR-875, INRA, F-31320 Castanet Tolosan, France*
   firstname.lastname@toulouse.inra.fr

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 par 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. Les modèles graphiques dits probabilistes ont pour objectif de maximiser le produit des fonctions. Une transformation directe existe entre les deux classes de modèles graphiques, qui peuvent également être modélisés en programmation linéaire en nombres entiers ou par un problème de satisfiabilité maximum en logique propositionnelle.*

Dans cet article, nous évaluons plusieurs logiciels de l'état de l'art en méthodes complètes d'optimisation sur une large collection de modèles graphiques déterministes et probabilistes issus de diverses compétitions MaxCSP 2008, Probabilistic Inference Challenge 2011, MaxSAT Evaluation 2013, MiniZinc Challenge 2012 & 2013, ainsi que des collections de problèmes de satisfaction de contraintes pondérées et en analyse d'images.

Au total, 3026 instances sont mises à disposition dans cinq formats et sept formulations. 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.

ABSTRACT. *By representing the constraints and objective function in factorized form, graphical models can concisely define various NP-hard optimization problems. They are therefore extensively used in several areas of computer science and artificial intelligence. Graphical models can be deterministic or stochastic, optimize a sum or product of local functions, defining a joint cost or probability distribution. Simple transformations exist between these two types of models, but also with MaxSAT or linear programming.*

In this paper, we report on a large comparison of exact solvers which are all state-of-the-art for their own target language. These solvers are all evaluated on deterministic and probabilistic

graphical models coming from the Probabilistic Inference Challenge 2011, the Computer Vision and Pattern Recognition OpenGM2 benchmark, the Weighted Partial MaxSAT Evaluation 2013, the MaxCSP 2008 Competition, the MiniZinc Challenge 2012 & 2013, and the CFLib (a library of Cost Function Networks).

All 3026 instances are made publicly available in five different formats and seven formulations. To our knowledge, this is the first evaluation that encompasses such a large set of related NP-complete optimization frameworks, despite their tight connections. The results show that a small number of evaluated solvers are able to perform well on multiple areas. By exploiting the variability and complementarity of solver performances, we show that a simple portfolio approach can be very effective. This portfolio won the last UAI Evaluation 2014 (MAP task).

MOTS-CLÉS : *modèle graphique, champ aléatoire de markov, problème de satisfaction de contraintes pondérées, programmation linéaire en nombres entiers, problème de satisfiabilité maximum en logique propositionnelle.*

KEYWORDS: *graphical model, markov random field, weighted constraint satisfaction problem, integer linear programming, MaxSAT.*

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 et à des requêtes d'optimisation combinatoire pour ces modèles. Ce cadre fait habituellement les hypothèses restrictives suivantes : faible arité des fonctions et taille restreinte des domaines permettant d'exprimer des fonctions quelconques par des tables.

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 (MaxSAT) a le même objectif. La programmation par contraintes (CP) peut également modéliser ces problèmes en introduisant des variables de coût (Petit *et al.*, 2000).

Les modèles graphiques probabilistes (PGM) (Koller, Friedman, 2009) 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 analyse 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* (Schlesinger, 1976 ; Koster, 1999 ; Globerson, Jaakkola, 2007) qui a plusieurs propriétés intéressantes.

Les modèles graphiques déterministes et probabilistes couvrent de très nombreux champs d'application de l'IA et de la RO pour lesquelles le problème à résoudre est un problème d'optimisation discrète. Plusieurs méthodes complètes [1] d'optimisation ont été développées dans différents langages de l'IA (CP, WCSP, MaxSAT, MRF) et de la RO (01LP), mais à notre connaissance il n'existe pas de comparaison de ces méthodes sur un ensemble de problèmes issus des langages de l'IA. En Section 2, nous présentons une synthèse des principaux langages développés en IA et RO pour représenter des modèles graphiques. En Section 3, nous proposons différentes reformulations pour passer d'un langage à l'autre. Enfin, en Section 4, nous comparons expérimentalement les différentes méthodes liées à ces langages sur une large collection d'instances.

1. Les méthodes sont comparées sur leur capacité à trouver des solutions optimales et prouver leur optimalité dans un temps limité.

Pour cela, nous avons collecté des instances de modèles graphiques déterministes et probabilistes issues de différentes sources incluant des compétitions CSP, MaxSAT, CP et PGM. 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 issues de l'état de l'art pour chacun de ces langages. Les résultats montrent que la meilleure méthode varie beaucoup en fonction du problème, suggérant des opportunités pour une approche portfolio d'algorithmes.

2. Langages d'optimisation discrète

Dans cette section, nous décrivons les différents langages d'optimisation discrète utilisés ensuite. Pour les modèles graphiques probabilistes, 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.

[MRF] Champ aléatoire de Markov

DÉFINITION 1. — *Un champ aléatoire de Markov discret*[2] *(Markov Random Field – MRF) est défini par une paire* (X, Φ) *avec* $X = \{x_1, \ldots, x_n\}$*, un ensemble de* n *variables aléatoires et* Φ*, un ensemble de* fonctions de potentiel*. Chaque variable* $x_i \in X$ *a un domaine fini* D_i *de valeurs qui peuvent lui être affectée. Une fonction de potentiel* $\phi_S \in \Phi$*, portant sur les variables* $S \subseteq X$*, est une fonction* $\phi_S : D_S \mapsto \mathbb{R} \cup \{\infty\}$*, 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 (Koller, Friedman, 2009). La probabilité $P(t)$ d'un n-uplet (tuple) donné $t \in D_X$ est définie par :

$$P(t) \propto \prod_{\phi_S \in \Phi} \exp(-\phi_S(t[S])) = \exp(-\sum_{\phi_S \in \Phi} \phi_S(t[S]))$$

avec $t[S]$, la projection d'un tuple t sur l'ensemble des variables S. La fonction ϕ_S est appelée fonction de potentiel additive ou aussi *énergie*, en relation avec la physique statistique. Alternativement, une fonction de potentiel multiplicative $\psi_S = \exp(-\phi_S(t[S])$ peut être utilisée.

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).

EXEMPLE 2. — 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 $\frac{\psi(a,a)}{\sum_{u \in D_x, v \in D_y} \psi(u,v)} = \frac{1}{3}$. □

2. Nous ne traitons pas le cas plus général de variables continues, voir par exemple (VanMarcke, 2010).

[WCSP] Problème de satisfaction de contraintes pondérées

Le problème de satisfaction de contraintes pondérées (Weighted CSP – WCSP) étend le formalisme CSP en remplaçant les contraintes par des fonctions de coût à valeurs entières positives (Meseguer *et al.*, 2006).

DÉFINITION 3. — *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.

[WPMS] Weighted Partial MaxSAT

Lorsque l'on se restreint à des domaines Booléens avec un langage de clauses pondérées, le problème WCSP devient un problème MaxSAT (Li, Manyà, 2009).

DÉFINITION 4. — *Un problème MaxSAT (Weighted Partial MaxSAT – WPMS) est défini par un ensemble de paires $\langle C, w \rangle$, avec C, une clause et $w \in \mathbb{N} \cup \{k\}$, 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 vaut k 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.

[01LP] Programmation linéaire à variables 0/1

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 (Jünger *et al.*, 2010).

[CP] Programmation par contraintes

Un problème de programmation par contraintes (CP) 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 (Rossi *et al.*, 2006).

3. Traduction entre les divers formalismes

Dans cette section, nous présentons l'encodage des différents modèles graphiques issus de l'IA/RO/CP 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.

[MRF] 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 [3].

La conversion d'une instance WCSP en une instance MRF avec potentiels multiplicatifs 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" [4].

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 5. — La fonction potentielle de l'exemple 2 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. □

[WPMS] Weighted Partial MaxSAT

Une instance MaxSAT 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* (Argelich *et al.*, 2008) et l'encodage de *tuple* (Bacchus, 2007).

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

3. A noter que des rationnels positifs sont aussi utilisés dans (M. Cooper *et al.*, 2010).
4. http://www.cs.huji.ac.il/project/PASCAL/fileFormat.php

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 6. — L'exemple 2 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 $(x \vee \neg y_b, 30), (x \vee \neg y_c, \infty), (\neg x \vee \neg y_a, 30), (\neg x \vee \neg y_c, \infty)$. □

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 (Bacchus, 2007). 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 7. — L'exemple 2 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` [5], qui inclut l'information du majorant k, de manière à préserver la propagation.

La complexité asymptotique des deux encodages est exprimée en terme du nombre total de tuples de coût 0 (t_0), k (t_k), ou autre (t_r) dans le problème. Pour le codage direct, la complexité est en $O(nd^2 + t_k + t_r)$, tandis que pour le codage de tuple, en $O(nd^2 + a(t_0 + t_r))$, avec n, le nombre de variables, d, la taille du plus grand domaine et a, l'arité maximum des fonctions de coût. Les termes constants cachés par cette notation O sont supérieurs pour le codage de tuple, qui a aussi un facteur linéaire supplémentaire a. Dans les benchmarks de nos expérimentations, nous observons qu'il y a beaucoup plus de tuples de coût nul que de coût infini ($t_0 >> t_k$).

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

[01LP] Programmation linéaire à variables 0/1

L'encodage d'une instance WCSP en 01LP[6] 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 8. — L'exemple 2 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 9. — L'exemple 2 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 (Koster, 1999). Il est équivalent à l'encodage en 01LP couramment utilisé pour les PGM dans le cas de fonctions d'arité au plus 2 (Koller, Friedman, 2009) (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 (Werner, 2007 ; Globerson, Jaakkola, 2007) et précédemment étudié par Schlesinger en analyse d'images pour le cas de grammaires de motifs 2D (Schlesinger, 1976). Ce polytope est intéressant pour plusieurs raisons. D'abord, le dual de ce polytope est identique à la cohérence d'arc optimale souple (OSAC) décrite dans (M. Cooper *et al.*, 2010) pour les WCSP. Il est aussi relié aux bornes produites par les algorithmes de propagation de mes-

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

sages (Werner, 2007 ; Globerson, Jaakkola, 2007) utilisés dans les MRFs. La borne produite par OSAC, équivalente à celle du polytope local, est toujours supérieure à celles produites par les autres cohérences d'arc souples comme EDAC (Larrosa *et al.*, 2005), à l'exception des possibles interactions avec la cohérence de noeud et la substituabilité (Givry *et al.*, 2013). De plus, les variables duales du polytope peuvent directement s'interpréter comme des quantités de coûts déplacés par les *transformations préservant l'équivalence* utilisées dans les cohérences d'arc souples (M. C. Cooper, Schiex, 2004). Ainsi, les cohérences d'arc souples, comme les algorithmes de propagation de messages, peuvent s'interpréter comme des algorithmes incrémentaux de descente de gradient par bloc dans le dual de ce polytope (Meltzer *et al.*, 2009). Enfin le polytope local (ou son dual) a récemment été montré comme étant très expressif au sens où n'importe quel programme linéaire peut être traduit en temps linéaire dans un modèle graphique dont le polytope local a le même optimum que le programme linéaire (Prusa, Werner, 2015).

[CP] Programmation par contraintes

Dans (Petit *et al.*, 2000), un encodage d'une instance WCSP en CP 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 MaxSAT, 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 CP `minizinc` (Nethercote *et al.*, 2007).

A l'inverse, traduire une instance CP 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` [7] (Hebrard *et al.*, 2010) lisant le format bas-niveau `flatzinc` [8] (Nethercote *et al.*, 2007). De plus, les contraintes globales ont été décomposées en fonctions de coût ($\{0, \infty\}$) d'arité 3 (Allouche, Bessiere *et al.*, 2012).

7. http://numberjack.ucc.ie/

8. Dans les expérimentations, une limite de 20 minutes a été imposée lors de la traduction de `minizinc` à `flatzinc`.

4. Comparaison de logiciels d'optimisation

4.1. Collection de benchmarks

Nous avons collecté 3026 instances provenant de différentes sources de modèles graphiques déterministes (WCSP, MaxSAT, CP) et probabilistes (MRF). Elles sont disponibles aux formats `uai`, `wcsp`, `wcnf`, `lp` et `minizinc` [9].

[MRF]: les instances probabilistes proviennent des compétitions *Uncertainty in Artificial Intelligence (UAI) 2008* (problème d'analyse de liaison génétique [10] (Favier *et al*., 2011), fonctions d'arité au plus 5) et *Probabilistic Inference Challenge (PIC) 2011* [11] (arité 2), au format natif `uai`. Les instances sont converties à 2 chiffres après la virgule via un passage au logarithme et un encodage en WCSP suivi d'une reconversion au format `uai` (extension *_digit2*) de manière à bien optimiser le même critère entre MRF et WCSP.

[CVPR]: d'autres instances probabilistes d'arité 2 et 3 sont issues d'un benchmark d'analyse d'images *OpenGM2 Computer Vision and Pattern Recognition (CVPR)* [12] (Kappes *et al*., 2015) au format natif `hdf5` et 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).

[WCSP]: CFLib [13] 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 (Larrosa *et al*., 2008), l'allocation de fréquence CELAR/GRAPH (Cabon *et al*., 1999), la correction d'erreur Mendélienne (Sanchez *et al*., 2008), la conception de protéines (Allouche, Traore *et al*., 2012) (avec une précision à 2 digits), la sélection de prises de vue du satellite SPOT5 (Bensana *et al*., 1999) et l'allocation d'entrepôts (Kratica *et al*., 2001 ; Larrosa *et al*., 2005).

[MaxCSP]: 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 & MaxCSP [14]. Ces instances au format natif `xcsp2.1/xml` ont été transformées en WCSP (MaxCSP)

---

9. http://genoweb.toulouse.inra.fr/~degivry/evalgm
10. http://graphmod.ics.uci.edu/uai08/Evaluation/Report/Benchmarks
11. http://www.cs.huji.ac.il/project/PASCAL, instances MAP/MPE à l'exclusion d'Alchemy, CSP, Promedas et ProteinProtein.
12. http://hci.iwr.uni-heidelberg.de/opengm2 à l'exclusion de Brain, Knott, MatchingStereo/ted-gm et ModularityClustering.
13. http://costfunction.org/benchmark
14. http://www.cril.univ-artois.fr/CPAI08/, http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html

tels que chaque tuple permis (resp. interdit) d'une contrainte a un coût nul (resp. unitaire) dans la fonction de coût correspondante. Nous fixons $k = 1000$.

[WPMS]: les instances MaxSAT proviennent des catégories Crafted (W)PMS (MIPLib & DIMACS Max Clique) et Industrial WPMS de la compétition *MaxSAT 2013 Evaluation* [15]. 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 MRF (resp. CP) imposent que les fonctions de coût soient de faible arité (resp. que les coûts tiennent sur 32 bits, excluant la catégorie WPMS/Upgradeability).

[CP]: nous avons pris des problèmes décomposables en WCSP des compétitions *MiniZinc Challenges 2012 & 2013* [16].

Dans la Table 1, nous indiquons le nombre d'instances par benchmark et les tailles compressée (gzip) pour les sept formulations. Dans le cas des problèmes issus de la CP, 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). Le format `uai` s'avère le plus compact du fait qu'il s'appuie sur un ordre implicite des tuples dont les coûts sont tous énumérés dans des tables en extension. La contre-partie est l'impossibilité d'exprimer des fonctions de coût de grande arité contenant peu de tuples non-nuls comme des clauses pondérés. L'encodage de tuple est le plus souvent plus volumineux que l'encodage direct, sauf pour MRF/CVPR `lp` où l'absence de coûts nuls rend l'encodage de tuple comparable en taille à l'encodage direct. Enfin les instances au format CP sont très compactes du fait des contraintes globales présentes dans le langage `minizinc` et traduites en larges tables dans les autres formats.

Tableau 1

Nombre d'instances et leur taille totale compressée (gzip) par format et par benchmark

Benchmark	Nb.	UAI	WCSP	LP(direct)	LP(tuple)	WCNF(direct)	WCNF(tuple)	MINIZINC
MRF	319	187MB	475MB	2.4G	2.0GB	518MB	2.9GB	473MB
CVPR	1461	430MB	557MB	9.8GB	11GB	3.0GB	15GB	N/A
WCSP	281	43MB	122MB	300MB	3.5GB	389MB	5.7GB	69MB
MaxCSP	503	13MB	24MB	311MB	660MB	73MB	999MB	29MB
WPMS	427	N/A [17]	387MB	433MB	N/A	717MB	N/A	631MB
CP	35	7.5MB	597MB	499MB	1.2GB	378MB	1.9GB	21KB
Total	3026	0.68G	2.2G	14G	18G	5G	27G	1.2G

Les plus grandes instances en nombre de variables proviennent des benchmarks WPMS et CVPR (presque 1 million de variables pour WPMS/TimeTabling et un demi-million pour CVPR/PhotoMontage et ColorSeg). De plus, MaxSAT contient des clauses portant sur un grand nombre de variables (580 dans Haplotyping). Pour les

15. http://maxsat.ia.udl.cat:81/13/benchmarks/

16. http://www.minizinc.org/challenge201{2|3}/results201{2|3}.html

17. Seuls WPMS/MaxClique et WPMS/MIPLib (sauf mod008) ont été traduits au format UAI pour un total de 8.8MB.

autres benchmarks, l'arité des fonctions reste faible entre 2 et 5. La connectivité du graphe des modèles graphiques probabilistes (MRF/CVPR) et WPMS est souvent très faible (utilisation le plus souvent d'une structure de grille pour CVPR). Cependant MRF/ObjectDetection, WCSP/ProteinDesign, MaxCSP/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 MaxCSP ne contiennent que des coûts 0/1. Tous les modèles graphiques déterministes, à l'exception de MaxCSP 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 discrète : `daoopt` [18] (réglage des paramètres à 1 heure (Otten *et al.*, 2012), sans amélioration des bornes par une variante de l'algorithme de *message passing*), vainqueur du challenge PIC 2011, `toulbar2` [19] (Hurley *et al.*, 2016) (réglage incluant la cohérence d'arc virtuelle (M. Cooper *et al.*, 2010) en prétraitement, l'élimination des valeurs dominées (Givry *et al.*, 2013) et une stratégie hybride d'exploration arborescente *meilleur-en-premier* (Allouche *et al.*, 2015)), vainqueur de MaxCSP 2008 et UAI 2010 & 2014, `maxhs` [20] (Davies, Bacchus, 2011 ; 2013), vainqueur de la catégorie crafted WPMS MaxSAT 2013, `gecode` [21], gagnant du challenge MiniZinc 2012 et enfin `cplex` 12.6.0.0 (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 avec une limite de temps CPU de 1 heure [22].

4.3. Résultats expérimentaux

18. https://github.com/lotten/daoopt version 1.1.2.
19. http://www.inra.fr/mia/T/toulbar2 version 0.9.8, paramètres *-A -V -dee -hbfs*.
20. http://www.maxhs.org version 2.51, pas de paramètre.
21. http://www.gecode.org/ version 4.4.0, mode *free search*.
22. Choix du paramètre *-pe parallel_smp 2* sur SUN Grid Engine pour assurer une utilisation maximale en demi-charge des noeuds du cluster.

TABLE 2 – Nombre de problèmes résolus en moins d'une heure (N/A si l'encodage du problème n'est pas possible pour des raisons mémoire ou de représentation des coûts en 32-bit). Entre parenthèses, temps CPU moyen en secondes pour les instances résolues ('-' sinon). En gras pour les meilleurs résultats. La première colonne contient le nom de la catégorie suivi par s: nb. d'instances, d: taille dom. max., a: arité max.

Problème/s/d/a	DAOOPT	TOULBAR2	CPLEX	CPLEX *tuple*	MAXHS	MAXHS *tuple*	GECODE
MRF/319/503/5 (UAI)	151 (584,39)	**226 (93,80)**	156 (111,88)	210 (82,18)	118 (172,27)	72 (98,68)	1 (1509,93)
DBN/108/2/2	60 (626,79)	**81 (192,42)**	65 (124,66)	69 (155,12)	38 (366,15)	2 (1748,65)	0 (-)
Grid/21/2/2	5 (1223,67)	0 (-)	**15 (120,90)**	1 (3354,21)	8 (557,01)	0 (-)	0 (-)
ImageAlignment/10/93/2	10 (754,96)	**10 (5,27)**	0 (-)	9 (88,41)	0 (-)	0 (-)	0 (-)
Linkage/22/7/5	17 (576,94)	14 (364,73)	16 (365,09)	**22 (21,99)**	20 (52,62)	20 (124,04)	0 (-)
ObjectDetection/37/21/2	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)
ProteinFolding/21/503/2	0 (-)	**21 (20,24)**	10 (169,28)	9 (176,17)	2 (268,51)	0 (-)	0 (-)
Segmentation/100/21/2	59 (460,33)	**100 (0,29)**	50 (0,06)	100 (3,35)	50 (7,38)	50 (22,54)	1 (1509,93)
CVPR/1461/20/3 (HDF5)	1274 (481,02)	**1340 (22,81)**	382 (179,96)	1332 (8,70)	483 (355,71)	1038 (58,83)	N/A
ChineseChars/100/2/2	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	N/A
ColorSeg/21/12/2	0 (-)	**15 (1340,56)**	0 (-)	5 (190,33)	0 (-)	0 (-)	N/A
GeomSurf/600/7/3	555 (509,01)	**600 (0,96)**	382 (179,96)	600 (2,89)	387 (183,53)	321 (63,15)	N/A
InPainting/4/4/2	0 (-)	**2 (325,72)**	0 (-)	1 (339,90)	0 (-)	0 (-)	N/A
Matching/4/20/2	4 (319,24)	**4 (3,20)**	0 (-)	3 (765,25)	0 (-)	0 (-)	N/A
MatchingStereo/2/20/2	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	N/A
ObjectSeg/5/8/2	0 (-)	4 (2292,28)	0 (-)	**5 (1057,88)**	0 (-)	0 (-)	N/A
PhotoMontage/2/7/2	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	N/A
ProteinInteraction/8/2/3	0 (-)	0 (-)	0 (-)	**3 (60,80)**	0 (-)	2 (1019,63)	N/A
SceneDecomp/715/8/2	715 (460,20)	**715 (0,07)**	0 (-)	715 (1,11)	96 (1049,83)	715 (54,20)	N/A
WCSP/281/300/3 (WCSP)	211 (768,93)	**256 (109,84)**	245 (33,85)	238 (34,00)	228 (14,91)	210 (121,20)	141 (224,77)
Auction/170/2/2	169 (663,04)	170 (93,10)	170 (0,03)	170 (0,14)	**170 (0,03)**	170 (121,16)	113 (231,55)
CELAR/16/44/2	4 (598,72)	**14 (279,00)**	0 (-)	3 (560,44)	0 (-)	0 (-)	0 (-)
Pedigree/10/28/3	4 (373,43)	**10 (10,58)**	5 (44,28)	9 (57,27)	10 (190,49)	6 (99,28)	0 (-)
ProteinDesign/10/198/2	4 (597,46)	**9 (13,40)**	0 (-)	7 (298,88)	0 (-)	4 (477,72)	0 (-)
SPOT5/20/4/3	6 (309,04)	4 (40,44)	**16 (22,99)**	12 (294,94)	6 (200,82)	5 (5,40)	0 (-)
Warehouse/55/300/2	24 (1752,42)	49 (163,23)	**54 (142,57)**	37 (6,46)	42 (6,78)	25 (92,83)	28 (197,39)
MaxCSP/503/50/2 (XCSP)	176 (603,56)	**398 (386,08)**	219 (152,73)	75 (876,84)	249 (76,21)	233 (538,93)	6 (115,39)
BlackHole/37/50/2	10 (222,19)	10 (0,08)	**30 (141,91)**	10 (2,22)	10 (0,30)	10 (2,78)	0 (-)
Coloring/22/6/2	17 (319,29)	17 (11,39)	**17 (7,14)**	16 (72,33)	14 (17,67)	14 (50,80)	4 (171,61)
Composed/80/10/2	26 (543,73)	**80 (0,13)**	80 (4,48)	37 (1667,07)	80 (79,81)	73 (1383,72)	0 (-)
EHI/200/7/2	0 (-)	**179 (773,86)**	0 (-)	0 (-)	1 (3078,96)	0 (-)	0 (-)
Geometric/100/20/2	92 (755,46)	**95 (134,57)**	65 (419,39)	0 (-)	89 (31,52)	84 (138,98)	0 (-)
Langford/4/29/2	2 (272,24)	**2 (0,12)**	2 (38,79)	1 (0,03)	2 (0,32)	2 (2,19)	2 (2,97)
QCP/60/9/2	29 (496,31)	15 (143,49)	25 (54,94)	11 (263,83)	**53 (121,82)**	50 (242,80)	0 (-)
WPMS/427/2/580 (WCNF)	11 (536,35)	197 (110,33)	269 (109,76)	N/A	**321 (168,67)**	N/A	28 (243,39)

Continue à la page suivante

Problème/s/d/a	DAOOPT	TOULBAR2	CPLEX	CPLEX *tuple*	MAXHS	MAXHS *tuple*	GECODE
Haplotyping/100/2/580	N/A	1 (784,32)	18 (679,90)	N/A	**44 (674,01)**	N/A	0 (-)
MIPLib/12/2/93	2 (365,31)	3 (102,39)	3 (49,85)	N/A	**3 (9,47)**	N/A	3 (28,61)
MaxClique/62/2/2	9 (574,36)	33 (209,07)	38 (229,33)	N/A	**40 (362,26)**	N/A	24 (280,38)
PackupWeighted/99/2/177	N/A	53 (167,82)	**99 (0,72)**	N/A	99 (7,14)	N/A	0 (-)
PlanningWithPref/29/2/372	N/A	7 (515,22)	11 (751,65)	N/A	**28 (65,82)**	N/A	1 (0,03)
TimeTabling/25/2/36	N/A	0 (-)	0 (-)	N/A	**7 (1020,73)**	N/A	0 (-)
Upgradeability/100/2/77	N/A	100 (12,43)	**100 (0,84)**	N/A	100 (2,73)	N/A	N/A
CP/35/*163/4* (MINIZINC)	9 (387,13)	16 (354,57)	2 (0,99)	7 (584,10)	18 (145,94)	14 (400,03)	**26 (138,55)**
AMaze/6/17/4	0 (-)	3 (279,71)	0 (-)	4 (998,46)	**6 (12,00)**	5 (161,25)	4 (176,91)
FastFood/6/*5*/2	1 (200,32)	1 (0,00)	1 (0,00)	1 (0,00)	1 (0,00)	1 (0,00)	**6 (14,22)**
Golomb/6/*163/3*	0 (-)	3 (44,97)	0 (-)	0 (-)	3 (117,34)	1 (78,01)	**6 (111,17)**
OnCallRostering/*5*/*89*/*4*	1 (253,25)	2 (27,27)	1 (1,98)	2 (47,44)	**3 (162,22)**	3 (362,19)	2 (75,13)
ParityLearning/7/20/4	7 (432,94)	7 (663,51)	0 (-)	0 (-)	5 (343,24)	4 (907,40)	**7 (248,10)**
VRP/5/100/4	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	**1 (255,45)**
Total	1832 (534,34)	**2433 (107,26)**	1273 (123,70)	1862 (57,35)	1417 (191,45)	1567 (143,45)	202 (219,37)
Nb. de 1ère place	0	**16 [1]**	7 [3]	3 [5]	9 [2]	0	4 [4]
Nb. de meilleures solutions	2209 [2]	**2562 [1]**	1355 [5]	1300 [6]	1626 [4]	1706 [3]	229[7]
Nb. de meilleures sol. uniques	57 [4]	88 [2]	43 [5]	**95 [1]**	80 [3]	1 [7]	13 [6]
Z-score (temps)	135,37 [6]	**57,84 [1]**	102,97 [3]	104,89 [4]	90,73 [2]	122,88 [5]	136,58 [7]
Z-score (coûts)	63,00 [3]	**26,25 [1]**	59,24 [2]	69,92 [4]	80,55 [5]	108,76 [7]	100,55 [6]
Score Borda	89,40 [5]	**182,50 [1]**	129,60 [2]	102,78 [4]	114,37 [3]	59,54 [7]	60,64 [6]
Score Borda (normalisé)	2,08 [5]	**4,24 [1]**	3,01 [2]	2,86 [3]	2,66 [4]	1,65 [7]	1,84 [6]

En Table 2 figure le nombre d'instances résolues en moins d'une heure, ainsi que le temps CPU moyen pour les instances entièrement résolues, sans les temps de conversion entre formats [23]. Nous donnons également à la fin de la table des résultats agrégés sur l'ensemble des catégories. Pour cela, deux Z-scores [24] sont calculés pour chaque instance à partir des temps CPU et des coûts des meilleures solutions trouvées à l'échéance. Lorsqu'un seul solveur est capable de résoudre une instance (resp. le solveur n'a pas réussi à résoudre l'instance), un score de -4 (resp. 4) est appliqué. Le Z-score moyen est calculé pour chaque catégorie et la somme des moyennes est donnée en Table 2.

Enfin, nous mesurons le score Borda qui combine temps CPU et coûts. Pour chaque instance et chaque paire de solveurs, une récompense entre $[0, 1]$ est accordée à chaque solveur de la façon suivante : si un solveur a trouvé une meilleure solution que l'autre solveur, il obtient une récompense de 1 (et 0 pour l'autre solveur). Lorsque le coût est identique, si t_0 et t_1 sont les temps CPU pour les deux solveurs numérotés 0 et

23. Des résultats détaillés par instance et des figures *cactus plot* se trouvent à http://genoweb.toulouse.inra.fr/~degivry/evalgm.

24. Le Z-score d'une valeur x dans un ensemble de valeurs vaut $\frac{x-\mu}{\sigma}$ où μ est la valeur moyenne de l'ensemble et σ son écart-type.

1, alors le solveur i reçoit une récompense $\frac{t_{|i-1|}}{t_0+t_1}$, favorisant le solveur le plus rapide. La moyenne des scores Borda est calculée pour chaque catégorie et la somme des moyennes est renseignée dans la table.

Ce dernier score est corrigé pour prendre en compte le fait que l'encodage de tuple et CP ne sont pas applicables dans les catégories WPMS et CVPR. Pour cela, nous avons normalisé le score Borda par le nombre de catégories applicables. La seule modification par rapport au score non-normalisé est l'ordre entre `cplex` avec l'encodage de tuple et `maxhs` avec l'encodage direct.

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 WPMS et `gecode` pour CP, les résultats montrent que certains solveurs comme `maxhs`, `cplex` et `toulbar2` marchent bien sur plusieurs classes, résolvant resp. 2043, 2313, and 2433 instances parmi 3026, gagnant la première place pour 9, 10 et 16 problèmes respectivement [25] parmi 43 catégories de problèmes. CVPR/ColorSeg est le problème avec l'espace de recherche le plus grand ($d^n = 4^{414720}$) résolu à l'optimum par `toulbar2`. MRF/ObjectDetection est le problème avec l'espace de recherche le plus petit entièrement non résolu ($d^n \leq 21^{60}$).

[MRF]: sur Linkage (Kishimoto, Marinescu, 2013), `cplex`, suivi par `maxhs`, obtient de très bons résultats, montrant sa capacité à traiter des fonctions de coût d'arité supérieure (5) avec des tuples interdits. De manière surprenante, l'encodage direct donne de meilleurs résultats sur le problème Grid ($n = 6400, d = 2$) pour `cplex` qui exploite un grand nombre de coupes *zero-half*. Les résultats de `daoopt` semblent décevants par rapport à ceux obtenus au challenge PIC 2011, une explication possible étant l'absence de reformulation de problème (code indisponible). Le solveur CP obtient 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 (Lee, Leung, 2012).

[CVPR]: sur le problème Scene Decomposition ($n = 208, d = 8$), qui utilise un modèle de *superpixel* (Kappes *et al.*, 2013), `toulbar2` résout toutes les 715 instances en $0,07$ secondes en moyenne comparé à $1,11$ secondes pour $\texttt{cplex}_t$. Les bonnes performances de `toulbar2` s'expliquent en grande partie par sa reformulation initiale via la cohérence d'arc virtuelle (VAC) (M. Cooper *et al.*, 2010). Sur ces problèmes, VAC produit un minorant proche de l'optimum plus rapidement que la relaxation linéaire sur le polytope local de l'encodage de tuple. Ce même encodage est celui qui marche le mieux pour `cplex`, en accord avec ce qui est communément établi pour les instances MRF. Bien que l'encodage de tuple fut toujours contre-productif pour un solveur WPMS sur les autres benchmarks (voir la colonne $\texttt{maxhs}_t$ dans la Table 2), ce n'est pas le cas pour CVPR (sauf pour GeomSurf-3).

25. En prenant le meilleur encodage à chaque fois pour `maxhs` et `cplex`.

[WCSP]: `toulbar2` domine clairement sur les problèmes CELAR ($n = 458, d = 44$), Pedigree ($n = 10017, d = 28$) et ProteinDesign ($n = 18, d = 198$), tandis que `cplex` avec l'encodage direct, suivi par `maxhs`, l'emporte sur les problèmes Auction ($n = 246, d = 2$) et Warehouse ($n = 1100, d = 300$) 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. GECODE obtient de bons résultats sur Auction et Warehouse, résolvant trois instances difficiles (*capmo-3-4-5* avec $n = 200, d = 100$).

[MaxCSP]: les bonnes performances de `maxhs` dans le benchmark MaxCSP peuvent s'expliquer par sa capacité à bien résoudre toutes les instances satisfiables (optimum à zéro) en particulier pour les problèmes Geometric ($n = 50, d = 20$) et QCP ($n = 264, d = 9$) 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 (Otten *et al.*, 2012), trouvant de bons majorants pour la phase suivante de recherche arborescente, spécialement sur les instances aléatoires comme EHI ($n = 315, d = 7$) et Geometric. `toulbar2` l'emporte sur quatre catégories, notamment EHI dû à sa nouvelle stratégie de recherche hybride (Allouche *et al.*, 2015) qui simule des redémarrages avec mémoire. Enfin, l'encodage de tuple s'avère toujours contre-productif pour ces problèmes.

[WPMS]: les clauses d'arité large interdisent d'appliquer l'encodage de tuple (du fait d'un trop grand nombre de tuples de coût nul) ni d'exprimer les problèmes en utilisant des tables pleines 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 PackupWeighted ($n = 25554, d = 2$), `cplex` est jusqu'à un ordre de grandeur plus rapide que `maxhs`. `gecode` s'avère le plus rapide à résoudre 11 instances MaxClique ($n = 3321, d = 2$), `maxhs` gagnant cette catégorie avec 40 instances résolues parmi 62.

[CP]: ces instances sont pour la plupart difficiles à traduire en fonctions de coût locales et avec de petits domaines (10 instances parmi 35 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 MaxSAT, `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 (Dechter, 1999), nécessitant ici un espace mémoire (529Mo, largeur induite inférieure à 25) inférieur à sa limite fixée par sa borne sur la largeur induite $i = 35$ et 4Go (Otten *et al.*, 2012).

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 fortement décomposable (faible largeur induite) et `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 analyse 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 un portfolio constitué de trois solveurs (`toulbar2`, `cplex` avec les deux encodages direct et de tuple, et `mplp2` (Sontag *et al*., 2012 ; 2008) [26]) et remporté ainsi la compétition UAI 2014 (Hurley *et al*., 2016) [27].

Nous espérons que la mise à disposition de cette collection de benchmarks, disponibles dans plusieurs formats, permettra d'enrichir les nombreuses compétitions en Intelligence Artificielle et Recherche Opérationnelle, conduisant à des solveurs plus robustes et à de nouvelles stratégies d'optimisation.

Remerciements

Nous sommes reconnaissant à la plateforme bioinformatique Genotoul de Toulouse pour l'accès au cluster et l'hébergement des benchmarks.

Bibliographie

Allouche D., Bessiere C., Boizumault P., Givry S., Gutierrez P., Loudni S. *et al*. (2012). Decomposing global cost functions. In *Proc. of AAAI.* Toronto, Canada.

Allouche D., Givry S. de, Katsirelos G., Schiex T., Zytnicki M. (2015). Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP. In *Proc. of CP*, p. 12–28. Cork, Ireland.

Allouche D., Traore S., Andre I., Givry S., Katsirelos G., Barbe S. *et al*. (2012). Computational protein design as a cost function network optimization problem. In *Proc. of CP*, p. 840–849. Quebec City, Canada.

Argelich J., Cabiscol A., Lynce I., Manya F. (2008). Encoding Max-CSP into partial Max-SAT. In *Proc. of ISMVL*, p. 106–111. Dallas, Texas, USA.

Bacchus F. (2007). GAC via unit propagation. In *Proc. of CP*, p. 133–147. Providence, USA.

Bensana E., Lemaitre M., Verfaillie G. (1999). Earth observation satellite management. *Constraints*, vol. 4, n° 3, p. 293-299.

26. http://cs.nyu.edu/~dsontag version 2 (2.10^{-7} gap thres.) a été employé pour la compétition mais les résultats obtenus par la suite ont montré qu'il était dominé par `toulbar2` aidé de VAC.
27. https://github.com/9thbit/uai-proteus

Cabon B., de Givry S., Lobjois L., Schiex T., Warners J. (1999). Radio link frequency assignment. *Constraints*, vol. 4, p. 79-89.

Cooper M., Givry S. de, Sanchez M., Schiex T., Zytnicki M., Werner T. (2010). Soft arc consistency revisited. *Artificial Intelligence*, vol. 174, p. 449-478.

Cooper M. C., Schiex T. (2004). Arc consistency for soft constraints. *Artificial Intelligence*, vol. 154, n° 1-2, p. 199-227.

Davies J., Bacchus F. (2011). Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proc. of CP*, p. 225–239. Perugia, Italy.

Davies J., Bacchus F. (2013). Exploiting the power of MIP solvers in MaxSAT. In *Proc. of SAT*, p. 166–181. Helsinki, Finland.

Dechter R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, vol. 113, n° 1–2, p. 41–85.

Favier A., Givry S., Legarra A., Schiex T. (2011). Pairwise decomposition for combinatorial optim. in graphical models. In *Proc. of IJCAI*, p. 2126–2132. Barcelona, Spain.

Givry S. de, Prestwich S., O'Sullivan B. (2013). Dead-end elimination for weighted CSP. In *Proc. of CP*, p. 263–272. Uppsala, Sweden.

Globerson A., Jaakkola T. (2007). Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *Proc. of NIPS*, p. 553–560. Vancouver, B.C., Canada.

Hebrard E., O'Mahony E., O'Sullivan B. (2010). Constraint Programming and Combinatorial Optimisation in Numberjack. In *Proc. of CP-AI-OR*, p. 181-185. Bologna, Italy.

Hurley B., O'Sullivan B., Allouche D., Katsirelos G., Schiex T., Zytnicki M. *et al.* (2016). Multi-Language Evaluation of Exact Solvers in Graphical Model Discrete Optimization. *Constraints*, vol. 21, n° 3, p. 413–434.

Jünger M. *et al.* (Eds.). (2010). *50 years of integer programming 1958-2008*. Springer.

Kappes J., Andres B., Hamprecht F., Schnorr C., Nowozin S., Batra D. *et al.* (2013). A comparative study of modern inference techniques for discrete energy minimization problem. In *Proc. of CVPR*. Portland, Oregon, USA.

Kappes J., Andres B., Hamprecht F., Schnorr C., Nowozin S., Batra D. *et al.* (2015). A comparative study of modern inference techniques for structured discrete energy minimization problems. *International Journal of Computer Vision*, vol. 115, n° 2, p. 155–184.

Kishimoto A., Marinescu R. (2013). Recursive best-first and/or search with overestimation for genetic linkage analysis. In *Proc. of CP workshop on constraint based methods for bioinformatics*, p. 17. Stockholm, Sweden.

Koller D., Friedman N. (2009). *Probabilistic graphical models: principles and techniques*. The MIT Press.

Koster A. (1999). *Frequency assignment: Models and algorithms*. Thèse de doctorat non publiée, University of Maastricht, The Netherlands.

Kratica J., Tosic D., Filipovic V., Ljubic I. (2001). Solving the simple plant location problem by genetic alg. *RAIRO*, vol. 35, n° 1, p. 127–142.

Larrosa J., de Givry S., Heras F., Zytnicki M. (2005). Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In *Proc. of IJCAI*, p. 84–89. Edinburgh, Scotland.

Larrosa J., Heras F., Givry S. de. (2008). A logical approach to efficient max-sat solving. *Artif. Intell.*, vol. 172, n° 2-3, p. 204-233.

Lee J., Leung K. (2012). Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction. *JAIR*, vol. 43, p. 257–292.

Li C. M., Manyà F. (2009). Maxsat. In *Handbook of satisfiability*, chap. 19. IOS Press.

Meltzer T., Globerson A., Weiss Y. (2009). Convergent message passing algorithms: a unifying view. In *Proc. of UAI*, p. 393–401. Montreal, Canada.

Meseguer P., Rossi F., Schiex T. (2006). Soft constraints processing. In F. Rossi, P. van Beek, T. Walsh (Eds.), *Handbook of constraint programming*, chap. 9. Elsevier.

Nethercote N., Stuckey P., Becket R., Brand S., Duck G., Tack G. (2007). MiniZinc: Towards a standard CP modelling language. In *Proc. of CP*, p. 529–543. Providence, USA.

Otten L., Ihler A., Kask K., Dechter R. (2012). Winning the PASCAL 2011 MAP challenge with enhanced AND/OR branch-and-bound. In *NIPS DISCML Workshop.* Lake Tahoe, NV, USA.

Petit T., Regin J., Bessiere C. (2000). Meta constraints on violations for over constrained problems. In *Proc. of ICTAI*, p. 358–365. Vancouver, BC, Canada.

Prusa D., Werner T. (2015). Universality of the local marginal polytope. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 37, n° 4, p. 898–904.

Rossi F., Beek P. van, Walsh T. (Eds.). (2006). *Handbook of constraint programming*. Elsevier.

Sanchez M., Givry S. de, Schiex T. (2008). Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, vol. 13, n° 1-2, p. 130-154.

Schlesinger M. (1976). Syntactic analysis of two-dimensional visual signals in noisy conditions. *Kibernetika*, vol. 4, p. 113–130.

Sontag D., Choe D., Li Y. (2012). Efficiently searching for frustrated cycles in MAP inference. In *Proc. of UAI*, p. 795–804. Catalina Island, California, USA.

Sontag D., Meltzer T., Globerson A., Weiss Y., Jaakkola T. (2008). Tightening LP relaxations for MAP using message-passing. In *Proc. of UAI*, p. 503-510. Helsinki, Finland.

VanMarcke E. (2010). *Random fields: Analysis and synthesis*. World Scientific Publishing Company.

Werner T. (2007). A linear programming approach to max-sum problem. *Pattern Analysis and Machine Intelligence*, vol. 29, n° 7, p. 1165–1179.