Onzièmes Journées Francophones de Programmation par Contraintes

JFPC 2015

Bordeaux, 22–24 juin 2015

JFPC 2015

Actes des Onzièmes Journées Francophones
de la Programmation par Contraintes

à l'initiative de l'Association Française pour la Programmation par Contraintes (AFPC)

Organisation

LABRI, Université de Bordeaux

Président des Journées

Laurent Simon, LABRI, Université de Bordeaux

Président du Comité de Programme

Gilles Audemard, CRIL, Université d'Artois

université de BORDEAUX

Comité d'organisation

Président	
Laurent Simon	LABRI, Bordeaux
Auriane Dantes	LABRI, Bordeaux
Katel Guérin	LABRI, Bordeaux

Comité de programme

Président	
Gilles Audemard	CRIL, Université d'Artois
David Allouche	INRA, Toulouse
Vincent Armant	The Insight Center for Data Analytics
Gilles Audemard	CRIL, Université d'Artois
Hadrien Cambazard	G-SCOP, Université Joseph Fourier, Grenoble
Fabien Chhel	LERIA, Université d'Angers
Thi-Bich-Hanh Dao	LIFO, Université d'Orléans
Sophie Demassey	CMA, MINES ParisTech
Gilles Dequen	MIS, Université de Picardie Jules Vernes
Jean-Guillaume Fages	Ecole des Mines de Nantes
Adrien Goeffon	LERIA, Universite d'Angers
Emmanuel Hebrard	LAAS, Toulouse
Jean Marie Lagniez	CRIL, Université d'Artois
Arnaud Lallouet	GREYC, Université de Caen - Basse Normandie
Olivier Lhomme	IBM
Xavier Lorca	Ecole des Mines de Nantes
Jean-Baptiste Mairy	Université Catholique de Louvain
Arnaud Malapert	I3S, Université de Nice-Sophia Antipolis
Jean-Noël Monette	Uppsala Universitet
Eric Monfroy	LINA, Université de Nantes
Samba Ndojh Ndiaye	Liris, Université de Lyon 1
Anastasia Paparrizou	University of Western Macedonia
Lionel Paris	LSIS, Aix-Marseille Université
Marie Pelleau	Université de Montréal
Olivier Roussel	CRIL, Université d'Artois
Laurent Simon	Labri, Bordeaux Institute of Technology
Sebastien Tabary	CRIL, Université d'Artois
Cyril Terrioux	LSIS, Aix-Marseille Université
Nadarajen Veerapen	Computing Science & Mathematics, University of Stirling, UK
Julien Vion	LAMIH, Université de Valenciennes et du Hainaut-Cambrésis
Philippe Vismara	LIRMM, Montpellier

Relecteurs additionnels

François Aubry, Anicet Bart, Clément Carbonnel, Laure Devendeville, Pierre Flener, Vinasetan Houndji, Tanguy Lapègue, Nadjib Lazaar, Christophe Lecoutre, Corinne Lucet, Younes Mechqrane, Farid Nouioua, Alexandre Papadopoulos, Cédric Piette.

Préface

Les Journées Francophones de Programmation par Contraintes (JFPC) constituent le principal congrès de la communauté francophone travaillant sur la programmation par contraintes (PPC), les problèmes de satisfaction de contraintes (CSP), de satisfiabilité (SAT) et de programmation logique avec contraintes (CLP). Les JFPC regroupent aussi des thématiques liées en recherche opérationnelle, sur les méta-heuristiques, l'analyse par intervalles, etc.

Les JFPC sont issues de la fusion des conférences JFPLC (Journées Francophones de Programmation en Logique et par Contraintes) nées en 1992 et des JNPC (Journées Nationales sur la résolution pratique des problèmes NP-complets) nées en 1994. Onzième édition de la série, les JFPC 2015 à Bordeaux succèdent à celles d'Angers (2014), Aix-en-Provence (2013), Toulouse (2012), Lyon (2011), Caen (2010), Orléans (2009), Nantes (2008), Rocquencourt (2007), Nîmes (2006) et Lens (2005).

L'objectif majeur des JFPC est de permettre aux scientifiques, industriels et étudiants francophones de se rencontrer, d'échanger et de présenter leurs travaux dans un cadre convivial. Cette année, les JFPC ont eu le plaisir de recevoir 45 soumissions provenant de Belgique, Canada, France, Irlande, Maroc, Tunisie, etc. Chaque article soumis a fait l'objet de relectures soignées par trois membres du comité de programme.

À l'issue des discussions menées sur la base des rapports, 40 articles ont été acceptés pour être présentés et publiés dans les actes, ce qui correspond à un taux d'acceptation de 88%. Sachant qu'une part conséquente des soumissions est constituée de versions en français d'articles de conférences et de journaux internationaux parmi les plus sélectifs du domaine, ce taux est révélateur de la qualité scientifique des JFPC.

Le comité de programme a réuni 30 scientifiques académiques et industriels, issus de 6 pays différents, représentant plus de 25 organismes. Je remercie très sincèrement les membres du comité et les relecteurs additionnels pour la pertinence et la qualité de leur travail, ainsi que les auteurs qui nous ont permis de proposer un programme scientifique très attractif.

En France, l'année prochaine va être riche en évènements scientifiques majeurs. En effet, la conférence CP aura lieu à Toulouse, la conférence SAT, se tiendra, elle, à Bordeaux et enfin le CRIL organisera l'école d'été d'ECAI à Lille. Ces trois évènements montrent le dynamisme de l'ensemble de notre communauté, dynamisme qui se ressent toutes les années durant les JFPC.

Pour terminer, je remercie l'Association Française pour la Programmation par Contraintes (AFPC) pour son investissement régulier en termes d'animation et de promotion de la communauté. Un grand merci à François Fages pour avoir accepter notre invitation et venir nous parler de ses travaux passionnants. Et enfin, je remercie vivement l'équipe d'organisation, les participants et les sponsors sans qui cet événement n'aurait pu avoir lieu.

Gilles Audemard
Président du comité de programme des JFPC 2015

Table des matières

Machinerie cellulaire et programmation biochimique : vers une informatique de la cellule

François Fages

Inria Paris-Rocquencourt
Francois.Fages@inria.fr

Résumé

La biologie des systèmes est un courant de recherche multi-disciplinaire qui cherche à comprendre les processus biologiques complexes en terme des interactions biochimiques à l'échelle de la cellule. Pour l'informaticien, la difficulté n'est pas tant dans le nombre d'espèces moléculaires et de leurs interactions, que dans la nature non conventionnelle du calcul biochimique qui est concurrent, distribué, partiellement analogique, et opéré par des systèmes de réactions acquis par l'évolution. Le pari de voir les cellules comme des machines, et les systèmes de réactions biochimiques comme des programmes, lance de nouveaux défis à l'informatique fondamentale et ouvre de nouvelles perspectives en biologie et en médecine. Dans cet exposé nous présenterons quelques concepts clés de cette démarche, à plusieurs reprises inspirée de la programmation logique avec contraintes, et l'illustrerons par un succès obtenu pour l'élucidation de la dynamique complexe de certaines voies de signalisation cellulaire.

Références

[1] Laurence Calzone, François Fages, and Sylvain Soliman. BIOCHAM : An environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14) :1805–1807, 2006.

[2] François Fages. Cells as machines : towards deciphering biochemical programs in the cell (invited talk). In Raja Natarajan, editor, *Proc. 10th International Conference on Distributed Computing and Internet Technology ICDCIT'14*, volume 8337 of *Lecture Notes in Computer Science*, pages 50–67. Springer-Verlag, 2014.

[3] François Fages, Steven Gay, and Sylvain Soliman. Inferring reaction systems from ordinary differential equations. *Theoretical Computer Science*, September 2014.

[4] François Fages and Pauline Traynard. Temporal logic modeling of dynamical behaviors : First-order patterns and solvers. In Luis Farinas del Cerro and Katsumi Inoue, editors, *Logical Modeling of Biological Systems*, chapter 8, pages 291–323. John Wiley & Sons, Inc., 2014.

[5] Steven Gay, François Fages, Thierry Martinez, Sylvain Soliman, and Christine Solnon. On the subgraph epimorphism problem. *Discrete Applied Mathematics*, 162 :214–228, January 2014.

[6] Domitille Heitzler, Guillaume Durand, Nathalie Gallay, Aurélien Rizk, Seungkirl Ahn, Jihee Kim, Jonathan D. Violin, Laurence Dupuy, Christophe Gauthier, Vincent Piketty, Pascale Crépieux, Anne Poupon, Frédérique Clément, François Fages, Robert J. Lefkowitz, and Eric Reiter. Competing G protein-coupled receptor kinases balance G protein and β-arrestin signaling. *Molecular Systems Biology*, 8(590), June 2012.

[7] Faten Nabli, Thierry Martinez, François Fages, and Sylvain Soliman. On enumerating minimal siphons in petri nets using CLP and SAT solvers : Theoretical and practical complexity. *Constraints*, pages 1–26, 2015.

[8] Jannis Uhlendorf, Agnés Miermont, Thierry Delaveau, Gilles Charvin, François Fages, Samuel Bottani, Gregory Batt, and Pascal Hersen. Long-term model predictive control of gene expression at the population and single-cell levels. *Proceedings of the National Academy of Sciences USA*, 109(35) :14271–14276, 2012.

Application efficace de la résolution pour Max-SAT sur les ensembles inconsistants

André Abramé et Djamal Habet

Aix Marseille Université, CNRS, ENSAM, Université de Toulon,
LSIS UMR 7296, 13397, Marseille, France
{andre.abrame,djamal.habet}@lsis.org

Résumé

L'application de la max-résolution dans les méthodes séparation et évaluation pour Max-SAT augmente le nombre de clauses de la formule initiale. Nous définissons un nouvel ordre d'application de cette règle qui vise à mieux contrôler cet inconvénient. Ce papier a été publié à la conférence CP 2014 [1].

Abstract

The application of the max-resolution rule in branch and bound solvers for Max-SAT increases the number of clauses in the formula. We introduce a new order of application of this rule to limit this drawback. This paper have been published in CP 2014 [1].

1 Introduction

Le problème Max-SAT consiste à trouver une interprétation qui maximise (minimise) le nombre de clauses satisfaites (falsifiées) d'une formule en forme normale conjonctive (CNF). Parmi les méthodes dédiées à la résolution de Max-SAT, les algorithmes de type *séparation et évaluation* ont montré leur efficacité sur les instances aléatoires et *crafted*. Un tel algorithme pour Max-SAT [4] explore l'ensemble de l'espace de recherche par un arbre de recherche. A chacun de ses nœuds, l'algorithme compare la meilleure solution déjà trouvée (la borne supérieure, BS) à une estimation de la meilleure solution atteignable sur la branche courante de l'arbre de recherche (la borne inférieure, BI). Si $BI \leq BS$ alors il est effectue un *retour-arrière*, sinon il sélectionne une variable non affectée (libre) et fixe sa valeur. L'un des composants importants d'un tel algorithme est l'estimation de la BI car elle est consommatrice en temps de calcul et influe fortement sur le nombre de nœuds explorés.

Le calcul de la BI consiste à réaliser une sous-estimation du nombre de sous-ensembles inconsistants (SI) disjoints à chaque nœud de l'arbre. Les SI sont détectés grâce à la propagation unitaire. Les étapes de propagation sont représentées par un graphe d'implications $G = (V, A)$ qui est orienté et acyclique. Les nœuds V représentent les littéraux propagés et les arcs A lient les causes des propagations (les littéraux falsifiés dans les clauses unitaires) à leurs conséquences (les littéraux propagés). Quand un conflit est détecté par la falsification de tous les littéraux d'une clause, un SI peut être construit en analysant la séquence de propagations qu'il l'a produit. Les SI sont alors traités pour assurer leur disjonctivité. L'un des traitements existants consiste à transformer les SI par la règle de la max-résolution suivante [2, 3] :

$$\frac{c_i = \{x, y_1, \ldots, y_s\},\ c_j = \{\overline{x}, z_1, \ldots, z_t\}}{cr = \{y_1, \ldots, y_s, z_1, \ldots, z_t\},\ cc_1,\ \ldots,\ cc_t,\ cc_{t+1},\ \ldots,\ cc_{t+s}}$$

c_i et c_j sont les clauses originelles, cr la résolvante et les clauses de compensation suivantes préservent l'équivalence de la formule : $cc_1 = \{x, y_1, \ldots, y_s, \overline{z}_1, z_2, \ldots, z_t\}$, $cc_2 = \{x, y_1, \ldots, y_s, \overline{z}_2, z_3, \ldots, z_t\}$, $\ldots$, $cc_t = \{x, y_1, \ldots, y_s, \overline{z}_t\}$, $cc_{t+1} = \{\overline{x}, z_1, \ldots, z_t, \overline{y}_1, y_2, \ldots, y_s\}$, $cc_{t+2} = \{\overline{x}, z_1, \ldots, z_t, \overline{y}_2, y_3, \ldots, y_s\}$, $\ldots$, $cc_{t+s} = \{\overline{x}, z_1, \ldots, z_t, \overline{y}_s\}$

Un SI est transformé en appliquant cette règle sur ses clauses. Aussi, la règle de la max-résolution est appliqués dans l'ordre inverse des propagations car elle ne peut s'appliquer sur un littéral que s'il réduit une seule clause (ou que les clauses qu'il réduit sont agrégées dans une seule résolvante). Enfin, la transformation d'un SI par cette règle produit des clauses de compensation qui peuvent augmenter rapidement la taille de la formule traitée.

2 Heuristique de la plus petite résolvante

Nous avons observé que le nombre et la taille des clauses de compensation varient selon l'ordre d'application de la règle de la max-résolution. Plus particulièrement, nous avons montré que pour deux variables adjacentes dans un graphe d'implications, l'application de cette règle sur la variable qui induit la plus petite résolvante produit toujours moins de clauses de compensation, qui sont aussi de plus petites tailles [1]. Sur cette base, nous avons proposé dans l'algorithme 1 une heuristique simple pour définir l'ordre de l'application de la règle de la max-résolution. Premièrement, le score de chaque variable $x \in var(V)$ [1] (ligne 2) est calculé comme suit : si x réduit une seule clause alors $score(x)$ est la taille de la résolvante intermédiaire, sinon $score(x) = \infty$. Deuxièmement, il sélectionne la variable avec le plus petit score (ligne 4) et applique la max-résolution entre son prédécesseur et son successeur (lignes 5-7). Il met alors à jour le graphe d'implications G (lignes 8) en remplaçant les clauses consommées par la résolvante intermédiaire (les arcs dupliqués sont supprimés) puis les scores des variables sont aussi mis à jour (ligne 9). Ce processus est répété tant que le graphe d'implications mis à jour n'est pas vide. La dernière résolvante produite est alors ajoutée à la formule. Nous appelons cette méthode l'heuristique de *la plus petite résolvante intermédiaire* (PRI).

Algorithme 1 : Transformation d'un SI par Max-SAT résolution selon l'heuristique PRI

Data : Une formule CNF Φ, une affectation I falsifiant une clause et le graphe d'implications correspondant $G = (V, E)$.
Result : La formule Φ transformée

```
1  begin
2      for x ∈ V do calculer le score de x;
3      while |V| > 0 do
4          l ← le littéral, tel que var(l) a le plus petit score
           dans V;
5          c1 ← prédécesseur de var(l);
6          c2 ← successeur de var(l);
7          cr ← max_sat_resolution(Φ,c1,x,c2);
8          G ← G_{<var(l)>};
9          Mettre à jour les scores des variables apparaissant
           dans c1 et c2;
10     Φ = Φ ∪ {cr};
```

3 Étude expérimentale

Nous avons implémenté l'heuristique PRI dans notre solveur de type séparation et évaluation AHMAXSAT. Dans la suite, AHMAXSAT désigne la variante utilisant l'ordre classique de l'application de la max-résolution et AHMAXSAT$^+$ correspond à celle utilisant PRI. Nous avons testé les deux variantes sur les instances aléatoires et *crafted* de la Max-Evaluation 2013, avec une limite de temps à 1800 secondes par instance. Les résultats obtenus montrent que AHMAXSAT$^+$ résout 3 instances de plus avec un gain de temps de 19% en moyenne. Le nombre et la taille des clauses de compensation sont significativement réduits (respectivement de 22% et 20% en moyenne). Par conséquent, l'estimation de la BI faite par AHMAXSAT$^+$ est plus précise et moins de nœuds sont développés.

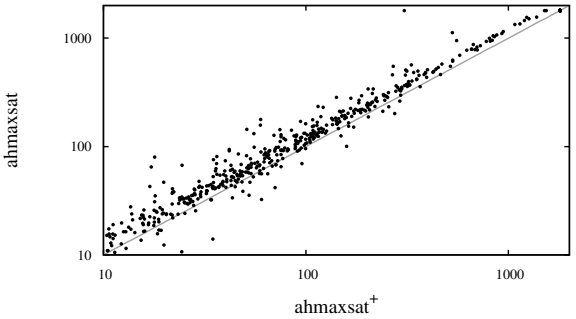


FIGURE 1 – Comparaison des temps de résolution (en secondes) d'AHMAXSAT et d'AHMAXSAT$^+$. Les axes sont en échelle logarithmique.

La Fig. 1 compare instance par instance les temps de résolution des deux variantes d'AHMAXSAT. Elle confirme l'observation précédente tout en indiquant que le gain est visible sur une la majorité du benchmark et qu'il n'est pas restreint à peu d'instances.

4 Conclusion

Nous avons introduit un nouvel ordre d'application de la règle de la max-résolution qui limite significativement l'augmentation de la taille de la formule. En perspective, nous étudierons l'impact de cette règle sur les autres caractéristique de la formule comme les propriétés de la propagation unitaire.

Références

[1] André Abramé and Djamal Habet. Efficient application of max-sat resolution on inconsistent subsets. In *CP 2015*, pages 92–107, 2014.

[2] María Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for max-sat. *Artificial Intelligence*, 171(8-9) :606–618, 2007.

[3] Javier Larrosa and Federico Heras. Resolution in max-sat and its relation to local consistency in weighted csps. In *IJCAI 2005*, pages 193–198, 2005.

[4] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for max-sat. *Journal of Artificial Intelligence Research*, 30 :321–359, 2007.

1. $var(V)$ (resp. $var(l)$) est l'ensemble des variables qui apparaissent dans V (resp. la variable concernée par l).

Maintenir et traiter toutes les sources de propagation unitaire dans les solveurs exacts pour Max-SAT

André Abramé et Djamal Habet

Aix Marseille Université, CNRS, ENSAM, Université de Toulon,
LSIS UMR 7296, 13397, Marseille, France
{andre.abrame,djamal.habet}@lsis.org

Résumé

Dans le cadre de la résolution de Max-SAT par les solveurs de type *séparation et évaluation*, nous introduisons un nouveau schéma de propagation unitaire qui prend en compte toutes les sources de propagation des variables. Nous montrons que ce schéma est plus adapté au contexte de Max-SAT et donnons un exemple de l'usage qui peut être fait des informations disponibles dans ce schéma. Ce travail a été publié à la conférence SoCS 2014 [1].

Abstract

In the context of Max-SAT solving by branch and bound algorithms, we introduce a new unit propagation scheme which considers all the propagation sources of the variables. We show that it is more relevant in the Max-SAT context and give an example of exploitation of the information that it contains. This work has been published at the SoCS 2014 conference [1].

1 Introduction

Le problème Max-SAT consiste à trouver une interprétation qui maximise (minimise) le nombre de clauses satisfaites (falsifiées) d'une formule en forme normale conjonctive (CNF). Parmi les méthodes dédiées à la résolution de Max-SAT, les algorithmes de type *séparation et évaluation* (S&E) ont montré leur efficacité sur les instances aléatoires et *crafted*. Ces algorithmes [4] explorent l'ensemble de l'espace de recherche en construisant un arbre de recherche. A chaque nœud de l'arbre, ils comparent la meilleure solution trouvée jusqu'à présent (la borne supérieure, BS) à une estimation de la meilleure solution atteignable dans la branche courante de l'arbre (la borne inférieure, BI). Si $BI \leq BS$ alors ils effectuent un *retour-arrière*, sinon ils sélectionnent une variable non affectée et fixent sa valeur. L'un des composants principaux des algorithmes S&E est l'estimation de la BI : elle est consommatrice en temps de calcul et influe fortement sur le nombre de nœuds explorés [4].

Le calcul de la BI consiste à réaliser une sous-estimation du nombre de sous-ensembles inconsistants (SI) disjoints à chaque nœud de l'arbre. Les SI sont détectés grâce à la propagation unitaire puis traités pour assurer leur caractère disjoint. Les traitements existants (suppression temporaire ou transformation par max-résolution [2, 3]) suppriment de la formule les clauses du SI pour la durée du calcul de la BI.

À notre connaissance, les implémentations existantes de la propagation unitaire, pour SAT et Max-SAT, se basent sur le schéma de la *première source de propagation* (PSP) : elles ne considèrent que la première clause unitaire (la source de propagation) ayant causé la propagation d'une variable donnée. Les autres sources de propagation sont alors satisfaites et donc ignorées. Dans ce schéma de propagation, lorsqu'une propagation est défaite (i.e. la variable correspondante est désaffectée), alors toutes les propagations qu'elle a engendrées doivent également être défaites pour pouvoir prendre en compte les autres sources de propagation précédemment ignorées. Ce schéma convient dans le cadre de SAT car les propagations ne sont défaites que lors des retours-arrières (et donc dans l'ordre chronologique inverse). Dans le cadre de Max-SAT, certaines clauses peuvent être supprimées lors du traitement des SI. Dans ce cas, les propagations engendrées par ces clauses doivent être défaites même si elles sont toujours valides (i.e. elles possèdent d'autres sources de

propagation). Cela peut conduire à défaire puis refaire inutilement les mêmes étapes de propagation unitaire.

2 Des sources multiples de propagation

Pour améliorer PSP, nous avons proposé un nouveau schéma de propagation que nous appelons *sources multiples de propagation* (SMP) [1]. Dans ce schéma, toutes les sources de propagation des variables sont considérées. Les variables propagées ne sont défaites que lorsqu'elles n'ont plus aucune source de propagation. Ainsi, les propagations peuvent être défaites dans un ordre non chronologique tout en limitant la redondance dans l'application de la propagation unitaire.

Dans ce nouveau schéma, nous modélisons les étapes de propagation par un *graphe d'implications plein* qui est un graphe ET/OU orienté. Les nœuds ET sont les clauses sources de propagation tandis que les nœuds OU sont les variables propagées [1]. Un tel graphe peut contenir des cycles. Pour éviter cela, nous attribuons à chaque variable propagée un niveau. La valeur de ce niveau est la longueur du plus long chemin (en nombre de nœuds) entre les nœuds racines du graphe et le nœud correspondant à la variable propagée. Ainsi, si une clause source à un niveau [1] supérieur à la variable qu'elle propage alors elle est ignorée car elle peut participer à l'apparition d'un cycle dans le graphe ET/OU.

Le schéma SMP peut aussi fournir des informations sur la structure de l'instance traitée, qui sont indisponibles dans le schéma PSP. Ces informations peuvent être exploitées notamment dans la construction des SI, selon des caractéristiques souhaitées, en choisissant les sources de propagation appropriées. Dans cette perspective, nous avons proposé une heuristique de choix de la source de propagation qui tente de minimiser la taille des SI, pour laisser le plus de clauses possibles disponibles aux propagations unitaires ultérieures [1]. Cela peut améliorer l'estimation de la BI et réduire ainsi le nombre de nœuds explorés. Cette heuristique, que nous appelons *petit ensemble inconsistant* (PEI), fonctionne comme suit : pour chaque variable propagée qui participe à un conflit (prise dans l'ordre inverse des propagations dans le graphe d'implications), PEI sélectionne la source de propagation qui contient le moins de nouvelles variables (à ajouter à celles figurant déjà dans le SI).

3 Étude expérimentale

Le schéma de propagation SMP est un composant central de notre solveur AHMAXSAT. En effet, une partie importante du code et des structures de données de AHMAXSAT dépendent fortement de ce schéma. Une implémentation efficace du schéma classique (PSP) dans AHMAXSAT nécessiterait une réécriture importante de son code. Pour ces raisons, nous évaluons l'impact de SMP sur le comportement de notre solveur de deux manières : (1) une évaluation de la part des propagations économisées grâce à SMP et (2) une comparaison de l'heuristique PSI à celle utilisée dans PSP.

L'étude est effectuée sur les instances aléatoires et *crafted* de *Max-SAT Evaluation 2013* avec un temps limite de 1800 secondes par instance. Les résultats obtenus montrent une réduction de 24% des étapes de propagation. Cette réduction varie entre 2.4% et 58% selon la classe d'instances considérée. Ce résultat est significatif au regard de l'importance des propagations unitaires dans les solveurs S&E.

Enfin, nous avons comparé l'impact de l'heuristique PSI à deux heuristiques basiques pour la construction des SI. La première choisit la première source de propagation disponible tandis que la deuxième sélectionne une source aléatoirement parmi celles construites. Les résultats obtenus montrent que l'heuristique PEI réduit le temps de calcul moyen de 10% environ.

4 Conclusion et perspectives

Nous avons introduit un nouveau schéma de propagation qui considère toutes les clauses causant la propagation des variables. Nous avons montré expérimentalement que ce nouveau schéma est plus approprié aux solveurs S&E pour Max-SAT. Nous avons illustré aussi une exploitation des informations contenues dans ce schéma afin d'influer sur les caractéristiques des ensembles inconsistants produits lors du calcul de la BI.

Nous pensons que l'intérêt de ce schéma dépasse le cadre de Max-SAT. Nous étudierons son usage dans les solveurs SAT complets tout en améliorant son utilisation pour Max-SAT.

Références

[1] A. Abramé and D. Habet. Maintaining and handling all unit propagation reasons in exact max-sat solvers. In *SOCS 2014*, pages 2–9, 2014.

[2] ML. Bonet, J. Levy, and F. Manyà. Resolution for max-sat. *Artificial Intelligence*, 171(8-9) :606–618, 2007.

[3] J. Larrosa and F. Heras. Resolution in max-sat and its relation to local consistency in weighted csps. In *IJCAI 2005*, pages 193–198, 2005.

[4] CM. Li, F. Manyà, and J. Planes. New inference rules for max-sat. *Journal of Artificial Intelligence Research*, 30 :321–359, 2007.

1. Le niveau d'une clause est le niveau maximum des variables qu'elle contient

Vers l'extension de l'apprentissage pour Max-SAT

André Abramé **Djamal Habet**

Aix Marseille Université, CNRS, ENSAM, Université de Toulon,
LSIS UMR 7296, 13397, Marseille, France.
{andre.abrame,djamal.habet}@lsis.org

Résumé

Un des composants les plus importants des solveurs Séparation et Évaluation (S&E) pour Max-SAT est l'estimation de la borne inférieure. À chaque nœud de l'arbre de recherche, elle consiste à détecter par des méthodes basées sur la propagation unitaire, puis à transformer les sous-ensembles inconsistants (SI) disjoints présents dans la formule simplifiée par l'interprétation partielle courante. Les solveurs récents appliquent une forme limitée d'apprentissage en mémorisant les transformations par max-résolution des SI qui correspondent à des motifs prédéfinis. Cependant, le calcul de la borne inférieure reste extrêmement redondant et les mêmes SI sont détectés et traités de nombreuses fois durant l'exploration de l'arbre de recherche. Nous nous intéressons dans cet article à l'augmentation de l'apprentissage réalisé par les solveurs S&E. Nous introduisons de nouveaux ensembles de motifs de clauses qui produisent des clauses résolvantes unitaires lorsqu'ils sont transformés par max-résolution. Nous étudions expérimentalement l'impact de l'apprentissage des transformations de ces motifs dans notre solveur ahmaxsat et analysons leurs effets sur sont comportement.

Abstract

One of the most critical components of Branch & Bound (BnB) solvers for Max-SAT is the estimation of the lower bound. At each node of the search tree, they detect the inconsistent subsets (IS) of the formula by unit propagation based methods and apply a treatment to them. The currently best performing Max-SAT BnB solvers perform a very little amount of memorization, thus the same IS may be detected and treated several times during the exploration of the search tree. We address in this paper the problem of increasing the learning performed by BnB solvers. We present new sets of clause patterns which produce unit resolvent clauses when they are transformed by max-resolution. We study experimentally the impact of these transformation' memorization in our solver ahmaxsat and we discuss their effects on the solver behavior.

1 Introduction

Le problème Max-SAT consiste à trouver, pour une formule CNF en entrée, une interprétation des variables Booléennes de la formule qui maximise (minimise) le nombre de clauses satisfaites (falsifiées). Dans la version pondérée de Max-SAT, un poids positif est associé à chaque clause et l'objectif est de maximiser (minimiser) la somme des poids des clauses satisfaites (falsifiées). Dans la version partielle de Max-SAT, les clauses de la formule sont divisées en deux catégories : les clauses dures et les clauses souples. L'objectif est alors de trouver une interprétation satisfaisant toutes les clauses dures et maximisant (minimisant) le nombre de clauses souples satisfaites (falsifiées). Enfin, la version partielle pondérée de Max-SAT est la combinaison des deux variantes précédentes. Par soucis de clarté, nous utilisons dans ce papier les notations associées aux problème Max-SAT (non-pondéré et non-partiel). Cependant, les résultats présentés peuvent être étendus aux autres variantes de Max-SAT.

Parmi les méthodes complètes pour résoudre le problème Max-SAT, les algorithmes de type Séparation et Évaluation (S&E) tels que WMAXSATZ [12, 15], AKMAXSAT [9] ou MINIMAXSAT [7, 8] ont prouvé leur efficacité, en particulier sur les instances aléatoires et *crafted*. Ces solveurs explorent l'ensemble de l'espace de recherche en construisant un arbre de recherche. À chaque nœud de cet arbre, ils calculent la Borne Inférieure (BI) en additionnant le nombre de clauses falsifiées par l'interprétation partielle courante et une (sous-)estimation de celles qui deviendront falsifiées si l'on étend cette interprétation. Puis, ils comparent BI à la meilleure solution trouvée jusqu'à présent (la Borne Supérieure, BS). Si BI$\geq$BS, alors il n'est pas possible de trouver une meilleure solution en étendant la branche courante de l'arbre de recherche et les solveurs réalisent un retour-arrière (*backtrack*). Le calcul de BI, et en particulier l'estimation des clauses qui deviendront falsifiées est un des composants les plus importants des sol-

veurs S&E : il est coûteux en temps de calcul et de sa qualité dépend grandement le nombre de nœuds qui seront explorés.

Les solveurs S&E récents estiment le nombre de clauses qui seront falsifiées en comptant les Sous-ensembles Inconsistants (SI) disjoints présents dans la formule à chaque nœud de l'arbre de recherche. Ils utilisent des méthodes basées sur la propagation unitaire pour détecter les inconsistances. Les étapes de propagation correspondantes sont ensuite analysées pour construire les SI. Chaque SI détecté doit ensuite être transformé pour qu'il ne soit compté qu'une fois. Deux méthodes de transformation sont utilisées par les solveurs S&E. Si un SI correspond à certains motifs prédéfinis, alors il est transformé par une série d'opérations de max-résolution [5, 6, 10, 11, 15] entre ses clauses, et la transformation est conservée dans la sous-partie de l'arbre de recherche. Ce traitement agit comme une forme (limitée) d'apprentissage. Les clauses des SI qui ne correspondent à aucun motif sont simplement supprimées de la formule ou transformées par max-résolution, et dans les deux cas les changements sont locaux au nœud courant (i.e. ils sont défaits avant la prochaine décision).

Nous proposons dans cet article d'augmenter la part d'apprentissage réalisée par les solveurs S&E. Nous nous intéressons en particulier aux sous-ensembles de la formule qui produisent, une fois transformés par max-résolution, des clauses résolvantes unitaires. L'apprentissage de ces transformations a un double avantage. Elle permet de réduire la redondance dans la détection et la transformation des SI et les clauses unitaires ainsi produites améliorent la capacité de la propagation unitaire à détecter d'autres SI. Nous définissons de nouveaux motifs correspondants à ces sous-ensembles, puis nous étudions expérimentalement l'impact de leurs apprentissages sur le comportement de notre solveur AHMAXSAT [1, 2, 3, 4] en faisant varier la taille de ces motifs et celle des clauses qu'ils contiennent. Les résultats obtenus montrent l'intérêt de cette approche et donnent des indices sur l'impact de l'apprentissage des transformations par max-résolution sur le comportement du solveur.

Ce papier est organisé comme suit. Nous donnons dans la section 2 les notations et définitions utilisées dans la suite de cet article et nous présentons la règle de la max-résolution. Dans la section 3, nous présentons les méthodes existantes pour la détection et la transformation des SI et nous rappelons les schémas d'apprentissage utilisés dans les solveurs S&E les plus récents. Nous introduisons de nouveaux ensembles de motifs dans la section 4 et discutons de leur détection. La section 5 est dédiée à l'étude empirique de l'inclusion de ces nouveaux motifs dans le schéma d'apprentissage. Nous concluons dans la section 6.

2 Préliminaires

Nous donnons dans cette section les principales définitions et notations utilisées dans la suite de cet article et nous présentons la règle de la max-résolution, qui est la pierre angulaire des méthodes d'apprentissage pour Max-SAT.

2.1 Définitions et Notations

Une formule Φ en forme normale conjonctive (CNF) définie sur une ensemble de variables propositionnelles $X = \{x_1, \ldots, x_n\}$ est une conjonction de clauses. Une clause c_j est une disjonction de littéraux et un littéral l_j est une variable x_i ou sa négation $\overline{x_i}$. Nous représenterons dans cet article les formules sous forme de multi-ensembles de clauses $\Phi = \{c_1, \ldots, c_m\}$ et les clauses sous forme d'ensembles de littéraux $c_j = \{l_1, \ldots, l_k\}$.

Une interprétation des variables de X peut être représentée comme un ensemble I de littéraux ne pouvant contenir à la fois un littéral et sa négation. Si x_i est affectée à *vrai* (resp. $faux$), alors $x_i \in I$ (resp. $\overline{x_i} \in I$). On dira qu'une interprétation I est complète si $|I| = |X|$ et qu'elle est partielle sinon.

Un littéral l est satisfait par une interprétation I si $l \in I$ et il est falsifié si $\overline{l} \in I$. Une variable qui n'apparaît ni positivement ni négativement dans I est dite non affectée. Une clause est satisfaite par I si au moins un de ses littéraux est satisfait et elle est falsifiée si tous ses littéraux sont falsifiés. Les clauses vides (notées $\square$) sont toujours falsifiées.

Résoudre une instance Φ du problème Max-SAT consiste à trouver une interprétation complète qui maximise (minimise) le nombre de clauses satisfaites (falsifiées) de Φ.

2.2 Règle de la max-résolution

La règle de la max-résolution [5, 6, 10] est une adaptation pour Max-SAT de la règle de la résolution pour SAT. Elle est définie comme suit :

$$\frac{c_i = \{x, y_1, \ldots, y_s\},\ c_j = \{\overline{x}, z_1, \ldots, z_t\}}{cr = \{y_1, \ldots, y_s, z_1, \ldots, z_t\},\ cc_1,\ \ldots,\ cc_t,\ cc_{t+1},\ \ldots,\ cc_{t+s}}$$

avec :

$$\begin{aligned}
cc_1 &= \{x, y_1, \ldots, y_s, \overline{z}_1, z_2, \ldots, z_t\},\\
cc_2 &= \{x, y_1, \ldots, y_s, \overline{z}_2, z_3, \ldots, z_t\},\\
&\ldots\\
cc_t &= \{x, y_1, \ldots, y_s, \overline{z}_t\},\\
cc_{t+1} &= \{\overline{x}, z_1, \ldots, z_t, \overline{y}_1, y_2, \ldots, y_s\},\\
cc_{t+2} &= \{\overline{x}, z_1, \ldots, z_t, \overline{y}_2, y_3, \ldots, y_s\},\\
&\ldots\\
cc_{t+s} &= \{\overline{x}, z_1, \ldots, z_t, \overline{y}_s\}
\end{aligned}$$

Les prémisses de la règle sont les clauses originelles c_i et c_j qui sont supprimées de la formule, tandis que les conclusions sont la clause résolvante *cr* et les clauses de compensation $cc_1, \ldots, cc_{t+s}$ ajoutées pour préserver l'équivalence de la formule.

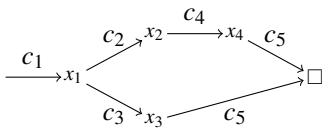
3 Apprentissage dans les solveurs S&E de l'état de l'art

À chaque nœud de l'arbre de recherche, les solveurs S&E calculent la borne inférieure (BI) en estimant le nombre de sous-ensembles inconsistants (SI) disjoints présents dans la formule simplifiée par l'interprétation courante. Nous présentons dans cette section les principales techniques de détection et de transformation des SI, puis nous rappelons les schémas d'apprentissage utilisés par les solveurs S&E les plus performants.

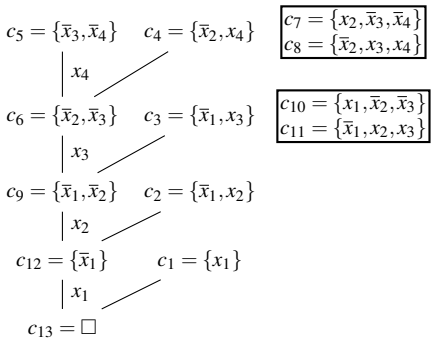
Les solveurs S&E récents utilisent des méthodes basées sur la propagation unitaire (PU) pour détecter les SI (plus précisément la propagation unitaire simulée (PUS) [13] et la méthodes des littéraux contradictoires (LC) [14]). Pour rappel, la propagation unitaire consiste à satisfaire les littéraux apparaissant dans des clauses unitaires jusqu'à ce qu'une clause vide (un conflit) soit produite ou qu'il n'y ait plus de clauses unitaires. Quand une clause vide est générée par PU, un sous-ensemble inconsistant peut être construit en analysant les étapes de propagation ayant mené au conflit. Ces étapes de propagation peuvent être représentées par un graphe d'implications [16], où les nœuds sont les littéraux propagés et les arcs connectent les littéraux falsifiés des clauses unitaires aux littéraux qu'elles propagent.

Une fois détecté, un SI peut être transformé en appliquant une série d'opérations de max-résolution entre ses clauses. Les clauses du SI sont supprimées de la formule tandis qu'une clause résolvante et un ensemble de clauses de compensation lui sont ajoutés. La clause résolvante est falsifiée par l'interprétation courante, donc PUS et LC ne sont plus nécessaires pour détecter l'inconsistance. En conservant ces transformations, les solveurs peuvent mémoriser les SI et éviter des détections et des transformations redondantes. L'exemple suivant illustre cette méthode de transformation.

Example 1 *Considérons la formule* $\Phi = \{c_1, \ldots, c_5\}$ *avec* $c_1 = \{x_1\}$, $c_2 = \{\bar{x}_1, x_2\}$, $c_3 = \{\bar{x}_1, x_3\}$, $c_4 = \{\bar{x}_2, x_4\}$ *et* $c_5 = \{\bar{x}_3, \bar{x}_4\}$. *L'application de PUS sur* Φ *conduit à la séquence de propagation* $\langle x_1@c_1, x_2@c_2, x_3@c_3, x_4@c_4\rangle$ *(x_1 est propagée par la clause c_1, puis x_2 par c_2, etc.). La figure 1a montre le graphe d'implications correspondant à cette séquence. La clause c_5 est alors vide et* Φ *est un sous-ensemble inconsistant. Sa transformation par max-résolution est réalisée comme suit. La max-résolution est d'abord appliquée entre les clauses c_5 et c_4 sur la variable x_4. La résolvante intermédiaire $c_6 = \{\bar{x}_2, \bar{x}_3\}$ est ajoutée à la formule ainsi que les clauses de compensation $c_7 = \{x_2, \bar{x}_3, \bar{x}_4\}$ et $c_8 = \{\bar{x}_2, x_3, x_4\}$. Les clauses originelles c_4 et c_5 sont supprimées de la formule. Puis, la max-résolution est appliquée entre la résolvante intermédiaire c_6 et la clause c_3 sur la variable x_3 et ainsi de suite. La figure 1b montre ces étapes de max-résolution (les cadres contiennent les clauses de compensation). On peut noter que cette méthode est proche de l'apprentissage de clause réalisé par les solveurs SAT modernes [16]. Après transformation complète, on obtient la formule* $\Phi' = \{\square, c_7, c_8, c_{10}, c_{11}\}$ *avec* $c_{10} = \{x_1, \bar{x}_2, \bar{x}_3\}$ *et* $c_{11} = \{\bar{x}_1, x_2, x_3\}$.



(a) Graphe d'implications



(b) Étapes de max-résolution

FIGURE 1 – Graphe d'implications et étapes de max-résolution appliquées sur la formule Φ dans l'exemple 1.

Cette méthode de transformation a cependant des inconvénient qui empêchent sa généralisation. Tout d'abord, l'ajout des résolvants et des clauses de compensation peut conduire à une augmentation rapide la taille de la formule. Ensuite, il a été observé expérimentalement que l'apprentissage de ces transformations pouvait mener à une dégradation de la qualité de l'estimation de la BI, entraînant l'augmentation du nombre de nœuds explorés de l'arbre de recherche.

Pour ces raisons, les solveurs S&E récents limitent l'apprentissage à la sous-partie courante de l'arbre de recherche (i.e. les changements sont défaits lors des retours-arrières). De plus, l'apprentissage n'est appliqué que pour la transformation des ensembles de clauses qui correspondent à un (ou une combinaison) des motifs suivants (avec au dessus les clauses des motifs et en dessous les clauses obtenues après application de la transformation) :

$$\frac{\{\{l_1,l_2\},\{l_1,\bar{l}_2\}\}}{\{\{l_1\}\}} \quad (1)$$

$$\frac{\{\{l_1,l_2\},\{l_1,l_3\},\{\bar{l}_2,\bar{l}_3\}\}}{\{\{l_1\},\{l_1,l_2,l_3\},\{\bar{l}_1,\bar{l}_2,\bar{l}_3\}\}} \quad (2)$$

$$\frac{\{\{l_1\},\{\bar{l}_1,l_2\},\{\bar{l}_2,l_3\},\ldots,\{\bar{l}_{k-1},l_k\},\{\bar{l}_k\}\}}{\{\square,\{l_1,\bar{l}_2\},\{l_2,\bar{l}_3\},\ldots,\{l_{k-1},\bar{l}_k\}\}} \quad (3)$$

Ces motifs ne couvrent pas nécessairement entièrement les SI. En particulier, l'application de la max-résolution sur les clauses des motifs 1 et 2 produit une clause résolvante unitaire. L'apprentissage de telles transformations réduit la redondance dans la détection et la transformation des SI. Les clauses unitaires ainsi produites peuvent également être utilisées dans la sous-partie de l'arbre de recherche pour réaliser davantage de propagations unitaires et donc de détecter potentiellement plus de SI.

4 Apprentissage étendu

Le calcul de la borne inférieure, basé sur l'estimation du nombre de sous-ensembles inconsistants (SI) disjoints, est un composant déterminant pour l'efficacité des solveurs S&E. En effet, il est très coûteux en temps de calcul et la qualité de l'estimation guide les retours-arrières et donc impacte le nombre de nœuds de l'arbre de recherche qui seront explorés. À cet égard, l'apprentissage semble un moyen naturel pour limiter le temps de calcul consacré au calcul de la BI en rendant la détection et le traitement des SI plus incrémental.

Nous donnons dans cette section une définition générale des motifs produisant après transformation des clauses résolvantes unitaires (*unit clause subset*, UCS). Nous motivons cette généralisation et montrons que ces motifs peuvent être détectés efficacement. Nous discutons également de la fréquence d'apparitions de ces motifs dans les SI des instances du benchmark présenté en section 5.

4.1 Unit Clause Subset (UCS)

Nous avons vu dans la section précédente que deux des motifs utilisés dans les schémas d'apprentissage produisent, après transformation, des clauses résolvantes unitaires. L'intérêt de la mémorisation de ces transformations est double. D'un coté, elle permet de limiter la redondance des étapes de propagation unitaire réalisées dans la sous-partie de l'arbre de recherche et d'un autre coté les clauses unitaires produites peuvent permettre de détecter plus de sous-ensembles inconsistants par propagation unitaire, et donc d'améliorer la qualité de la borne inférieure.

Nous proposons dans ce chapitre d'étendre l'apprentissage réalisé par les solveurs en prenant en compte de nouveaux motifs produisant, après transformation par max-résolution, des clauses résolvantes unitaires. Ces motifs peuvent être définis formellement comme suit.

Definition 1 (*unit clause subsets*, UCS) *Considérons une formule CNF Φ. Un sous-ensemble produisant des clauses résolvantes unitaires est un ensemble $\{c_1,\ldots,c_k\} \subset \Phi$ avec $\forall i \in \{1,\ldots,k\}$, $|c_i| > 1$ et tel qu'il existe une séquence d'étapes de max-résolution sur les clauses $c_1,\ldots,c_k$ qui produit une clause résolvante unitaire. Nous notons k-UCS l'ensemble des motifs correspondant aux UCS de taille k.*

Example 2 *Ci-dessous les motifs de l'ensemble 3-UCS :*

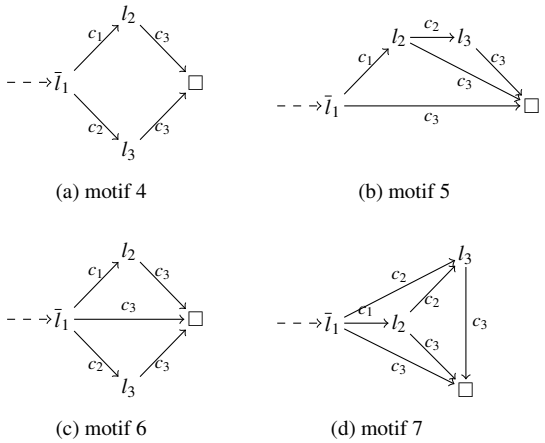
$$\frac{\{\{l_1,l_2\},\{l_1,l_3\},\{\overline{l_2},\overline{l_3}\}\}}{\{\{l_1\},\{l_1,l_2,l_3\},\{\overline{l_1},\overline{l_2},\overline{l_3}\}\}} \quad (4)$$

$$\frac{\{\{l_1,l_2\},\{\overline{l_2},l_3\},\{l_1,\overline{l_2},\overline{l_3}\}\}}{\{\{l_1\},\{\overline{l_1},\overline{l_2},l_3\}\}} \quad (5)$$

$$\frac{\{\{l_1,l_2\},\{l_1,l_3\},\{l_1,\overline{l_2},\overline{l_3}\}\}}{\{\{l_1\},\{l_1,l_2,l_3\}\}} \quad (6)$$

$$\frac{\{\{l_1,l_2\},\{l_1,\overline{l_2},l_3\},\{l_1,\overline{l_2},\overline{l_3}\}\}}{\{\{l_1\}\}} \quad (7)$$

Les graphes d'implications présentés dans la figure 2 décrivent des exemples d'étapes de propagation qui peuvent menées à la détection de ces motifs.



(a) motif 4 (b) motif 5

(c) motif 6 (d) motif 7

FIGURE 2 – Graphes d'implications décrivant des exemples de séquences de propagations unitaires permettant de détecter les motifs de l'ensemble des 3-UCS présentés dans l'exemple 2. On suppose que le littéral $\overline{l_1}$ a déjà été propagé et que les clauses de chacun des motifs sont nommées c_1, c_2 et c_3.

Puisque l'un des objectifs de l'apprentissage des transformations des UCS est d'accroître les affectations réalisées par propagation unitaire (PUS ou LC) lors du calcul de la borne inférieure, nous ne considérons pas les sous-ensembles contenant des clauses unitaires. Dans le meilleur des cas (s'ils contiennent une seule clause unitaire), la transformation de tels sous-ensembles laisserait inchangé le nombre de clauses unitaires présentes dans la formule. Dans le pire des cas (s'ils contiennent plus d'une clause unitaire), la formule transformée contiendrait moins de clauses unitaires que l'originel. Le nombre d'affectations faites par propagation unitaire, et par conséquent le nombre de SI détectés, peuvent alors être réduits.

Nous faisons une distinction entre les motifs des ensembles k-UCS selon la taille de leurs clauses. Nous notons par k^b-UCS le sous-ensemble de k-UCS composé des motifs qui ne contiennent que des clauses binaires et k^t-UCS le sous-ensemble composé des motifs contenant au moins une clause ternaire. Pour tout $j > 0$, nous noterons par $\{k_1, k_2, \ldots, k_j\}$-UCS l'union des ensembles k_1-UCS, k_2-UCS, ..., k_j-UCS.

On peut noter que les motifs 1 et 2 appartiennent respectivement aux ensembles 2-UCS et 3^b-UCS.

4.2 Détection et fréquence d'apparitions des k-UCS

Les motifs des k-UCS sont facilement détectables lors de l'analyse du graphe d'implications réalisée pour construire les sous-ensembles inconsistants. En effet, les clauses situées entre le conflit et le premier point d'implication unique (*first unit implication point*, FUIP) [16] produisent lorsqu'elles sont transformées par max-résolution une résolvante unitaire. Il suffit donc de compter le nombre et la taille de ces clauses pour savoir s'il on est en présence d'un UCS valide. Cela ne change pas la complexité de la procédure d'analyse des conflits et le surcoût en terme de temps de calcul est négligeable.

Nous avons mesuré le taux d'apparitions des motifs k-UCS dans les sous-ensembles inconsistants détectés par notre solveur AHMAXSAT sur les instances du benchmark présenté en section 5. Les résultats sont présentés dans la table 1. Il est intéressant d'observer qu'en moyenne, les motifs utilisés dans le schéma le plus répandu d'apprentissage (2-UCS et 3^b-UCS) sont présents dans moins de 5% des SI. En moyenne, des UCS sont détectés dans plus 57% des sous-ensembles inconsistants trouvés par AHMAXSAT, ce qui montre que la prise en compte des UCS dans le schéma d'apprentissage peut potentiellement réduire sensiblement les redondances dans le calcul de la borne inférieure.

On peut remarquer que ces valeurs varient fortement selon les classes d'instances. Par exemple, le taux moyen d'apparitions des UCS est inférieur à 3,5% sur les instances

TABLE 1 – Pourcentages d'apparitions des motifs k-UCS dans les SI détectés par AHMAXSAT.

Classes d'instances			2	3^b	3^t	4^b	4^t	5^b	5^t	> 5
ms	crafted	bipartite	0,05	0,52	0,03	0,22	0,08	42,19	0,29	42,13
		maxcut	0,06	15,71	0,04	4,15	9,45	5,24	6,73	21,49
		set-covering	-	-	-	-	-	-	-	-
	random	highgirth	1,47	0,79	0,49	0,58	0,8	0,34	1,15	23,54
		max2sat	0,12	3	0,05	20,03	0,59	10,63	2,57	41,98
		max3sat	1,23	3,11	1,45	4,5	4,3	1,92	6,79	38,02
		min2sat	0,39	2,76	0,11	16,93	0,29	12,18	0,9	43,2
pms	crafted	frb	0,11	12,63	0,03	0,02	25,38	2,52	3,05	19,16
		job-shop	-	-	-	-	-	-	-	-
		maxclique	0,09	10,25	0,05	0,03	9,33	3,76	0,95	17,33
		maxone	1,89	0,32	0,56	0,25	0,82	0,15	0,47	5,3
		min-enc/kbtree	0,21	0,74	0,15	2,74	0,45	1,34	1,48	20,41
		pseudo/miplib	0,42	1,17	0,26	0,11	0,67	0,18	5	9,48
		reversi	0,51	2,62	0,37	0,43	2,3	1,45	1,34	29,39
		scheduling	-	-	-	-	-	-	-	-
	random	min2sat	0,57	0,23	0,33	0,12	0,28	6,66	0,28	60,09
		min3sat	0,18	2,33	0,08	0,15	0,66	11,69	0,33	51,22
		pmax2sat	0,96	8,03	0,14	15,92	1,04	5,88	3,43	29,53
		pmax3sat/hi	1,11	1,9	0,65	3,12	1,78	1,85	3,17	37,25
wpms	crafted	auction	0,35	8,73	0,24	0,03	15,82	0,16	3,26	3,79
		CSG	0,16	3,66	0,01	0,05	0,41	>0,01	0,16	1,53
		frb	0,11	13,46	0,04	0,03	21,05	3,04	2,69	19,37
		min-enc	0,29	0,94	0,47	1,4	0,66	0,3	1,17	16,45
		pseudo/miplib	0,27	0,5	0,8	0,28	2,6	0,12	1,64	35,8
		ramsey	0,29	5,18	0,21	0,07	1,49	0,63	1,82	11,42
		random-net	0,07	1,4	0,2	>0,01	8,48	>0,01	18,1	4,02
		set-covering	0,89	0,27	0,7	0,02	0,58	0,02	0,35	0,6
		wmaxcut	0,08	13,74	0,05	0,44	12,65	6,21	8,56	22,48
	random	wmax2sat	0,09	3,35	0,03	21,57	0,47	11,33	2,92	40,64
		wmax3sat	0,89	2,02	0,97	3,11	3,31	1,43	5,96	38,7
		wpmax2sat	0,44	5,21	0,08	12,91	0,67	6,76	2,71	32,77
		wpmax3sat/hi	0,81	1,12	0,48	1,88	1,28	1,17	2,38	33,9
Ensemble			0,5	4,43	0,29	6,84	3,24	8,13	2,54	31,09

de la classe wpms/crafted/set-covering tandis qu'il est supérieur à 85% sur celles de la classe ms/crafted/bipartite. On peut aussi noter que les taux d'apparitions de chaque sous-ensemble des UCS varient également d'une classe d'instances à une autre. Cela met en exergue les différences structurelles entre les instances de ces classes.

5 Évaluation empirique de l'apprentissage des UCS

Nous présentons dans cette section une évaluation expérimentale de l'impact sur les performances de notre solveur AHMAXSAT de l'ajout des motifs des k-UCS à son schéma d'apprentissage. Tous les tests sont effectués sur les instances aléatoires et *crafted* du benchmark de la *Max-SAT Evaluation (MSE) 2014* [1]. Nous n'incluons aucune instance issue de problèmes industriels. Même si les résultats présentés ici peuvent être étendus à ces instances, les solveurs S&E (notre solveur AHMAXSAT y compris, et à l'excep-

1. `www.maxsat.udl.cat/14/benchmarks/index.html`

tion de MINIMAXSAT [8]) ne gèrent pas efficacement les instances industrielles de très grande taille [2].

Nous avons implémenté le schéma d'apprentissage étendu grâce aux UCS dans notre solveur AHMAXSAT [3] [1, 2, 3, 4]. Les tests sont effectués sur des machines équipées de processeurs Intel Xeon 2.4 Ghz, de 24 Gb de mémoire vive et de systèmes d'exploitation GNU/Linux. Le temps de résolution est limité à 1800 secondes et la mémoire à 3.5 Gb pour chaque instance traitée (comme lors des MSE).

En partant d'une variante d'AHMAXSAT utilisant les motifs du schéma d'apprentissage classique ($\{2,3^b\}$-UCS), nous ajoutons successivement les motifs de l'ensemble 3^t-UCS, puis ceux de 4-UCS et ceux de 5-UCS. Des résultats préliminaires (non présentés dans cet article) suggèrent que l'ajout des motifs des ensembles k-UCS avec $k > 5$ ont un impact négatif sur les performances du solveur. Toutes les variantes incluent également dans leur schéma d'apprentissage le motif 3 qui n'est pas un UCS. Les résultats sont présentés dans la table 2. Pour chaque variante d'AHMAXSAT, nous présentons le nombre d'instances résolues (colonnes S), le nombre moyen de nœuds explorés (colonnes D), le temps moyen de résolution (colonnes T), le nombre moyen de propagations effectuées à chaque nœud de l'arbre de recherche (colonnes P/D) et le nombre moyen de sous-ensembles inconsistants détectés à chaque nœud de l'arbre de recherche (colonnes □/D). Le rôle de ces deux derniers indicateurs est de mesurer la réduction des redondances dans l'application de la propagation unitaire et dans la détection des SI. Ils peuvent également indiquer une dégradation de la précision de l'estimation de la borne inférieure.

$\{2,3^t\}$-UCS vs. $\{2,3\}$-UCS Comme on pouvait s'y attendre, l'ajout des 3^t-UCS aux motifs utilisés par les solveurs existants (les $\{2,3^b\}$-UCS) ne change pas fondamentalement le comportement d'AHMAXSAT puisque les sous-ensembles inconsistants ne correspondent que très rarement à ces motifs (0,29% en moyenne, cf. table 1). Sur les classes d'instances où le taux d'apparitions des 3^t-UCS n'est pas nul, on peut observer une légère réduction du nombre moyen de propagations et de SI détectés par nœud de l'arbre de recherche (colonnes P/D et □/D) qui se traduit par une légère amélioration de la vitesse d'exploration de l'arbre de recherche (non représentée dans la table 2). Le nombre moyen de nœuds explorés et le temps moyen de résolution sont aussi réduits sur ces classes d'instances (colonnes D et T).

Par exemple, sur la classe ms/random/max3sat, qui a le plus fort pourcentage d'apparitions des 3^t-UCS dans les SI (1,45% en moyenne), on peut voir que le nombre moyen de propagations et de SI détectés par nœud sont réduits respectivement de 1% et 2%. La vitesse moyenne d'exploration de l'arbre de recherche passe de 1306,5 nœuds par secondes à 1326,5 (+1,1%). De plus, le nombre de nœuds explorés est réduit de 5,1%. La combinaison de ces deux effets produit une réduction du temps moyen de résolution de 6,1%.

Ces résultats confirment les bénéfices attendus de l'apprentissage des transformations des UCS :

1. l'augmentation de la vitesse d'exploration de l'arbre de recherche due à la réduction de la redondance dans le calcul de la borne inférieure, et
2. la réduction du nombre de nœuds explorés due à l'amélioration de la qualité de la borne inférieure induite par l'augmentation du nombre de clauses unitaires dans la formule.

On peut noter également que ce second effet masque en partie l'impact du premier dans les statistiques que nous présentons. La réduction de la redondance fait baisser les valeurs des indicateurs P/D et □/D tandis qu'à l'inverse, l'augmentation du nombre de clauses unitaires les augmente.

Si l'on considère les résultats sur l'ensemble du benchmark, le nombre moyen de nœuds explorés et le temps moyen de résolution sont réduits respectivement de 0,8% et 1%. Comme nous l'avons dit précédemment, la faible ampleur de ces résultats est à mettre en rapport avec la faible fréquence d'apparitions des motifs de l'ensemble 3^t-UCS dans les SI détectés par AHMAXSAT.

$\{2,3\}$-UCS vs. $\{2,3,4\}$-UCS Si l'on ajoute les motifs de l'ensemble 4-UCS au schéma d'apprentissage, les nombres moyens de propagations et de SI détectés par décisions sont réduits significativement (respectivement -5,3% et -17,1%). Comme la quantité d'apprentissage réalisée par le solveur augmente, on peut s'attendre à une légère réduction de ces deux valeurs, mais pas dans de telles proportions. Ces valeurs indiquent probablement une perte dans la capacité de détection des sous-ensembles inconsistants, et donc une réduction de la qualité de la borne inférieure. On peut remarquer que sur les classes d'instances présentant les plus fortes réductions de ces indicateurs, comme ms/random/max2sat ou wpms/random/wmax2sat, le nombre moyen de nœuds explorés augmente fortement (respectivement +472,1% et +210,7%) ce qui vient confirmer notre interprétation. Sur l'ensemble du benchmark, 10 instances de moins sont résolues et le temps moyen de résolution est augmenté de 19,1%.

Il est intéressant d'observer que les résultats varient fortement selon les classes d'instances. La perte de performances ne survient que sur les classes d'instances ayant de forts pourcentages d'apparitions des 4^b-UCS. A l'inverse, sur les classes d'instances ayant de faibles pourcentages d'apparitions des 4^b-UCS, comme pms/crafted/frb ou

2. www.maxsat.udl.cat/14/results/index.html
3. www.lsis.org/habetd/Djamal_Habet/MaxSAT.html

TABLE 2 – Comparaison de l'impact des schémas d'apprentissage basés sur les motifs $\{2, 3^b\}$-UCS et $\{2, 3\}$-UCS. Les deux premières colonnes montrent les classes d'instances et le nombre d'instances par classe. Les colonnes S, D, T, P/D et □/D donnent respectivement le nombre d'instances résolues, le nombre moyens de nœuds explorés, le temps moyen d'exécution, le nombre moyen de propagations faites par nœud de l'arbre de recherche et le nombre moyen de sous-ensembles inconsistants détectés par nœud de l'arbre de recherche. Les colonnes marquées avec une étoile ne prennent en compte que les instances résolues par toutes les variantes.

	Classes d'instances	#	$\{2,3^b\}$-UCS					$\{2,3\}$-UCS					$\{2,3,4\}$-UCS					$\{2,3,4^r\}$-UCS					$\{2,3,4^r,5\}$-UCS					$\{2,3,4^r,5^r\}$-UCS				
			S	D*	T*	P/D*	□/D*	S	D*	T*	P/D*	□/D*	S	D*	T*	P/D*	□/D*	S	D*	T*	P/D*	□/D*	S	D*	T*	P/D*	□/D*	S	D*	T*	P/D*	□/D*
ms	crafted	bipartite	100	100	33658	88,3	1135,3	73,9	100	33726	88,4	1135,4	74	100	33485	87,9	1135,6	74	100	33518	90,7	1135,8	74	98	184518	318,9	687,8	30	100	32328	84	1132,4	73,4
		maxcut	67	56	241129	63,1	146,1	22	56	241129	63	146,1	22	56	164939	42,1	109,2	15,5	56	167956	48,5	142,9	20,1	56	192076	49	127,8	17,3	56	171380	44,7	135,2	18,3
		set-covering	10	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-
	random	highgirth	82	7	4563195	1045,3	43,2	2,5	7	4649532	1064,2	43,1	2,5	7	4636028	1065,9	43,1	2,5	7	4647438	1121,2	43	2,5	6	4680189	1086,2	43,1	2,5	7	4650016	1072,9	43,3	2,5
		max2sat	100	100	30406	62,1	926,9	48,6	100	30484	62,4	927,5	48,7	93	174399	291,3	699,6	31,6	100	29710	62,4	926,3	48,6	90	206837	334,3	681	29,8	100	28292	55,6	890,8	44,4
		max3sat	100	100	372213	284,9	231,6	24,5	100	353221	267,5	229,9	24	100	355812	250,4	213,1	21,1	100	325667	242,3	221,3	22,3	100	367432	249,3	202,4	19,6	100	340968	231,2	205,8	20,1
		min2sat	96	96	1067	2,7	1097,5	35,9	96	1067	2,7	1097,9	35,9	96	1259	2,5	843	19,9	96	1065	2,7	1096,3	35,8	96	3689	7	844,6	20,9	96	1046	2,6	1096,1	35,7
pms	crafted	frb	25	5	1364465	333,5	49,8	6,9	5	1364465	333,8	49,8	6,9	5	801715	205,2	47,1	5,2	5	804408	209,3	47	5,2	5	687551	176,4	43,9	4,6	5	585949	154,2	46,1	4,8
		job-shop	3	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-
		maxclique	158	133	25474	28,7	88,3	7,5	133	25474	27,8	88,3	7,5	133	22426	30,5	86,2	6,4	133	22340	28,4	86,1	6,4	133	26735	33,9	231,7	5,8	133	23107	30,2	86,1	6,1
		maxone	140	109	148070	31,2	718,8	23,2	109	148071	31,4	718,6	23,2	107	133744	28,5	717,2	23	109	133749	28,8	718,6	23,1	108	133742	28,3	1105,5	27,4	109	133747	28,7	718,7	23,1
		min-enc/kbtree	42	34	202579	500,8	1035,3	63,7	34	208641	515,8	1035,1	63,7	34	202738	488,4	1027,5	62,8	34	209900	507,2	1032,2	63,5	35	205645	494	979,4	60,8	34	197998	476,2	1037	63,5
		pseudo/miplib	4	2	268	0	11,8	1,8	2	268	0	11,8	1,8	2	268	0	11,8	1,8	2	268	0	11,8	1,8	2	288	0	4726,2	17	2	288	0	11,8	1,8
		reversi	44	8	4679	129,9	2596,2	23,9	8	4679	129,3	2596,2	23,9	8	4719	130	2648,1	24,5	8	4736	128,1	2637,7	24,5	8	4708	132,2	5135,3	34,6	8	4555	129,2	2907,4	27
		scheduling	5	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-
	random	min2sat	60	58	5271	68,7	6055,5	51,9	58	5278	68,6	6055,5	51,9	58	5281	69,4	6056,5	51,9	58	5283	69,2	6056,5	51,9	51	12161	161,9	5658,3	42,1	58	5283	69,4	6056,4	51,9
		min3sat	60	58	67532	181,2	1199,2	27,6	58	67587	181,6	1199,3	27,6	58	66974	180,7	1195,6	27,3	58	66887	186,2	1195,9	27,3	52	187338	451,5	1018,6	23,1	58	66917	181,2	1195,8	27,3
		pmax2sat	60	60	1045	3,5	944,4	74,6	60	1066	3,6	946,5	74,5	60	2781	8,9	863,4	62	60	1037	3,5	931,4	72,4	60	2257	6,8	838,2	58,6	60	1021	3,2	878,4	65,8
		pmax3sat/hi	30	30	89897	65	242,4	16,4	30	89236	64,6	242,2	16,4	30	86400	60,9	236,2	15,6	30	86932	62,2	240,4	16,1	30	86893	60,3	229,8	17,4	30	84944	60,1	235,8	15,6
wpms	crafted	auction	40	40	467409	123,5	143,7	66,4	40	467518	124,1	143,4	66,3	40	309981	111,1	127,1	73,9	40	310614	112	126,8	73,7	40	310322	118,4	111,8	72,3	40	310021	115,5	116,7	74,6
		CSG	10	4	54378	432,4	16,9	0	4	54378	453	16,9	0	4	54648	432	15,4	0	4	54378	445,4	16,9	0	4	54813	431,7	14,2	1,2	4	54813	452,3	16,4	0
		frb	34	14	492276	128,6	40,4	6	14	492276	125	40,4	6	14	290082	82,4	36,1	4,4	14	291044	78,7	36	4,4	14	248789	71,5	35,4	4,1	14	212369	58,7	36,5	4,3
		min-enc	74	64	21340	109,1	8805,1	78,2	64	21340	109,2	8805,1	78,2	64	24009	117,8	8561,6	73,5	64	20522	107,6	8805,4	76,8	64	21554	111,6	10838,8	74,8	64	20101	108,8	8826,3	77,1
		pseudo/miplib	12	4	1500	159,4	2859	162,8	4	1500	159,3	2859	162,8	4	1438	155,8	2602,5	166,4	4	1493	157,6	2906,2	169,2	4	1496	159,5	158,5	32,9	4	1505	158,8	2852,7	166,4
		ramsey	15	4	158529	44,2	7	2,1	4	158529	44,3	7	2,1	4	158131	44,3	7	2,1	4	158161	44,6	7	2,1	4	157496	45	23,3	7,2	4	157559	44,8	7	2,2
		random-net	32	1	1253990	1069,5	-	-	1	1253990	1078,6	-	-	1	1563328	1366,2	-	-	1	1563328	1360,2	-	-	3	335161	240	-	-	3	335161	237,5	-	-
		set-covering	45	25	3754	42,3	453,7	46,8	25	3754	42,5	453,7	46,8	25	3754	42,7	453,7	46,8	25	3754	44,1	453,7	46,8	25	3764	42,9	501,7	51	25	3754	46,4	453,7	46,8
		wmaxcut	48	46	53152	85,9	597,1	122,5	46	53214	86,6	597,1	122,5	46	28518	51,4	434,5	97,3	46	28679	53,7	599,9	109,6	46	28737	51,1	613,4	99,6	46	26985	49,4	582,4	97,9
	random	wmax2sat	120	120	4192	48,1	2595,5	279,3	120	4188	47,8	2596,1	279,5	119	13014	131,6	2307,9	182	120	4182	48	2591,6	277,6	118	16188	170,2	2226,1	169,7	120	3810	43,2	2538,3	262
		wmax3sat	40	40	41332	115,2	612,4	106,7	40	40387	112,2	611,2	105,8	40	38980	107	598,2	100,1	40	37939	106,2	605,1	102,7	40	38221	103	579,6	95,3	40	36836	99,5	591,3	97,7
		wpmax2sat	90	90	575	5,9	1172,3	209,2	90	573	5,7	1170,7	210	90	841	12	1322,3	209,3	90	575	5,7	1156,1	211,8	90	777	10,5	1270,6	197	90	575	5,5	1144,6	205,5
		wpmax3sat/hi	30	30	31591	88,4	656,6	64,6	30	31464	87,8	656,8	64,5	30	30751	85,6	653,7	63,3	30	31004	87,5	656,2	64,1	30	30705	85,1	660,9	64,4	30	30451	84,1	654	63,4
		Ensemble	1776	1438	112580	92,7	1344,7	77,4	1438	111719	91,8	1344,6	77,4	1428	108233	109,3	1273,3	64,2	1438	96202	87,3	1340,7	76,5	1412	126692	144,3	1347,1	58,6	1440	94407	82,7	1328,8	73,3

wpms/crafted/frb, le temps moyen de résolution est réduit (-39,5% et -34,1% respectivement). Ces résultats suggèrent que la mémorisation des transformations des SI correspondant aux motifs 4^b-UCS impacte négativement et dans des proportions importantes les performances de notre solveur.

$\{2,3\}$-UCS vs. $\{2,3,4^t\}$-UCS Si l'on ignore les 4^b-UCS et que l'on ne mémorise que les transformations des SI correspondant aux motifs $\{2,3,4^t\}$-UCS, le nombre moyen de nœuds explorés et le temps moyen de résolution sur l'ensemble du benchmark sont réduits respectivement de 13,9% et de 4,9%. On peut observer que les nombres moyens de propagation et de SI détectés par nœud sont légèrement réduits (respectivement -0,3% et -1,2%).

$\{2,3,4^t\}$-UCS vs. $\{2,3,4^t,5\}$-UCS Lorsqu'on ajoute les motifs 5-UCS au schéma d'apprentissage d'AHMAXSAT, on observe le même comportement qu'avec les motifs 4-UCS. Sur l'ensemble du benchmark, le nombre moyen de SI détectés par nœud est réduit significativement (-23,3%). En conséquence, le nombre moyen de nœuds explorés augmente (+31,7%), 26 instances de moins sont résolues et le temps moyen de résolution est plus élevé (+65,3%). Comme précédemment, la perte de performances est particulièrement élevée sur les classes d'instances ayant un fort pourcentage d'apparitions des motifs 5^t-UCS dans les SI, comme les ms/crafted/bipartite, ms/random/max2sat, pms/random/min3sat ou wpms/random/wmax2sat où l'augmentation du temps moyen de résolution est respectivement de 251,5%, 435,7%, 142,5% et 254,6%.

$\{2,3,4^t\}$-UCS vs. $\{2,3,4^t,5^t\}$-UCS Comme précédemment, si l'on ignore les motifs 5^b-UCS et qu'on ne mémorise que les motifs $\{2,3,4^t,5^t\}$-UCS, 2 instances de plus sont résolues, le nombre moyen de nœuds et le temps moyen de résolution sont réduits respectivement de 1,9% et 5,3%.

Bilan L'ajout des motifs $\{3^t,4^t,5^t\}$-UCS au schéma d'apprentissage de notre solveur AHMAXSAT permet de résoudre 2 instances de plus. Le nombre moyen de nœuds explorés et le temps moyen de résolution sont réduits de 16,2% et 11,8% respectivement. Les figures 3a et 3b comparent instance par instance respectivement le temps de résolution et le nombre de nœuds explorés d'AHMAXSAT$\{2,3^b\}$-UCS et d'AHMAXSAT$\{2,3,4^t,5^t\}$-UCS. On peut observer que le gain est régulièrement réparti sur l'ensemble du benchmark.

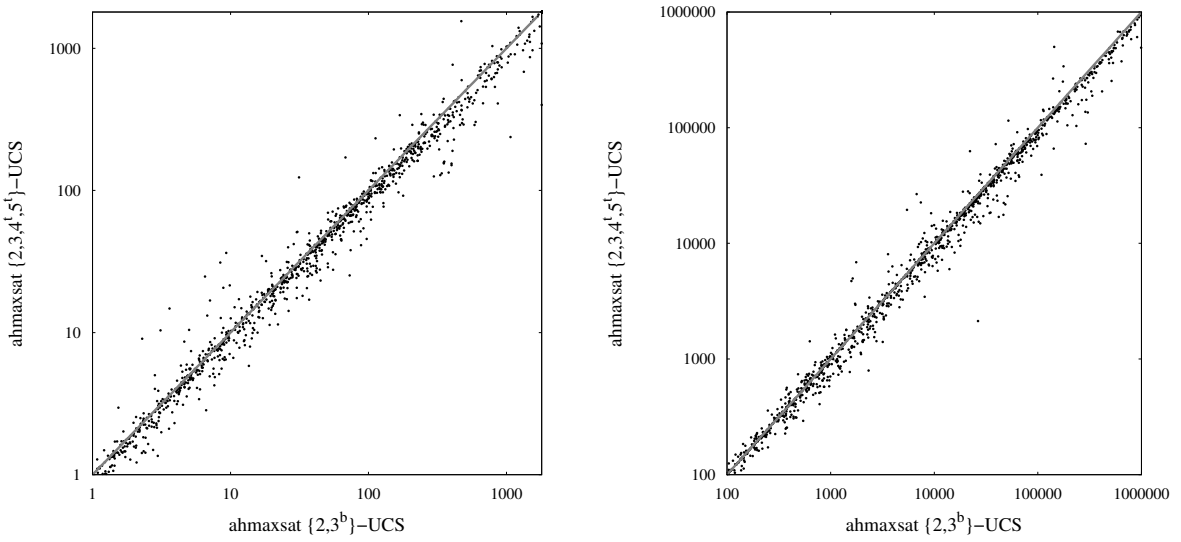
Enfin, la figure 4 compare les temps de résolutions cumulés de AHMAXSAT$\{2,3^b\}$-UCS et AHMAXSAT$\{2,3,4^t,5^t\}$-UCS avec ceux de trois des solveurs S&E les plus performants : AKMAXSAT [9] et deux variantes de WMAXSATZ [15, 12]. On peut voir que le gain apporté par l'ajout des motifs $\{3^t,4^t,5^t\}$-UCS au schéma d'apprentissage d'AHMAXSAT est significatif. Sur les instances considérées, les deux variantes de notre solveur sont nettement supérieures aux autres solveurs S&E de l'état de l'art. Ces résultats sont conformes à ceux de la dernière *Max-SAT Evaluation*, où AHMAXSAT (avec l'apprentissage étendu) a été classé premier dans trois des neuf catégories de l'évaluation.

Discussion Il est communément admis que l'apprentissage de la transformation des SI par max-résolution peut dans certains cas réduire la qualité de l'estimation de la BI. Cependant, à notre connaissance, les causes de ce comportement n'ont jamais été décrites clairement. L'étude empirique que nous avons réalisée montre que l'apprentissage de la transformation des clauses relatives à certains motifs (les 4^b-UCS et les 5^b-UCS) semble particulièrement affecté par ce phénomène. De plus, les statistiques détaillées présentées précédemment montrent une réduction du nombre de propagations et de SI détectés à chaque nœud de l'arbre de recherche et donc une augmentation du nombre de nœuds explorés. En effet, si le nombre d'étapes de propagation est réduit, alors moins de SI seront détectés et l'estimation de la BI sera moins précise. Par conséquent, les retours-arrières seront effectués plus bas dans l'arbre de recherche et davantage de nœuds seront explorés. Nous illustrons dans l'exemple ci-dessous l'impact de la transformation des clauses d'un motif 4^b-UCS sur la propagation unitaire.

Example 3 *Considérons le sous-ensemble $\Phi'' = \{c_2,c_3,c_4,c_5\}$ de la formule Φ introduite dans l'exemple 1. Si l'on ajoute la clause $c_{14} = \{\bar{x}_1,x_4\}$ à Φ'', il y a deux UCS possibles : $\psi_1 = \{c_3,c_5,c_{14}\}$ et $\psi_2 = \{c_2,c_3,c_4,c_5\}$ qui sont respectivement un 3^b-UCS et un 4^b-UCS.*

Si ψ_1 est transformé par max-résolution, on obtient la formule $\Phi^{(3)} = \{c_2,c_4,c_{15},c_{16},c_{17}\}$ avec $c_{15} = \{\bar{x}_1\}$, $c_{16} = \{\bar{x}_1,x_3,x_4\}$ et $c_{17} = \{x_1,\bar{x}_3,\bar{x}_4\}$. Dans $\Phi^{(3)}$, l'affectation de x_1 à vrai conduit à la séquence de propagations $\langle x_2@c_2, x_4@c_4\rangle$. La clause c_{15} est falsifiée tandis que c_{16} et c_{17} sont satisfaites.

Si ψ_2 est transformé par max-résolution, on obtient la formule $\Phi^{(4)} = \{c_{12},c_7,c_8,c_{10},c_{11},c_{14}\}$ (cette transformation est décrite dans l'exemple 1). L'affectation de x_1 à vrai conduit, dans $\Phi^{(4)}$, à la séquence de propagations unitaires $\langle x_4@c_{14}\rangle$. La clause c_{12} est falsifiée, c_8 et c_{10} sont satisfaites et c_7 et c_{11} sont réduites. Il n'y a plus de clauses unitaires et x_2 ne peut pas être propagée. Il est intéressant d'observer que les deux clauses réduites $c_7 = \{x_2,\bar{x}_3\}$ et $c_{11} = \{x_2,x_3\}$ pourraient mener à la propagation de x_2 si l'on appliquait la max-résolution entre elles sur la variable x_3. Mais la propagation unitaire seule ne permet pas de propager x_2.



(a) Temps de résolution (en secondes).

(b) Nombre de nœuds.

FIGURE 3 – Comparaison du nombre de nœuds explorés et du temps de résolution d'AHMAXSAT avec les ensembles de motifs $\{2,3^b\}$-UCS et $\{2,3,4^t,5^t\}$-UCS. Chaque point représente une instance. Tous les axes sont en échelle logarithmique.

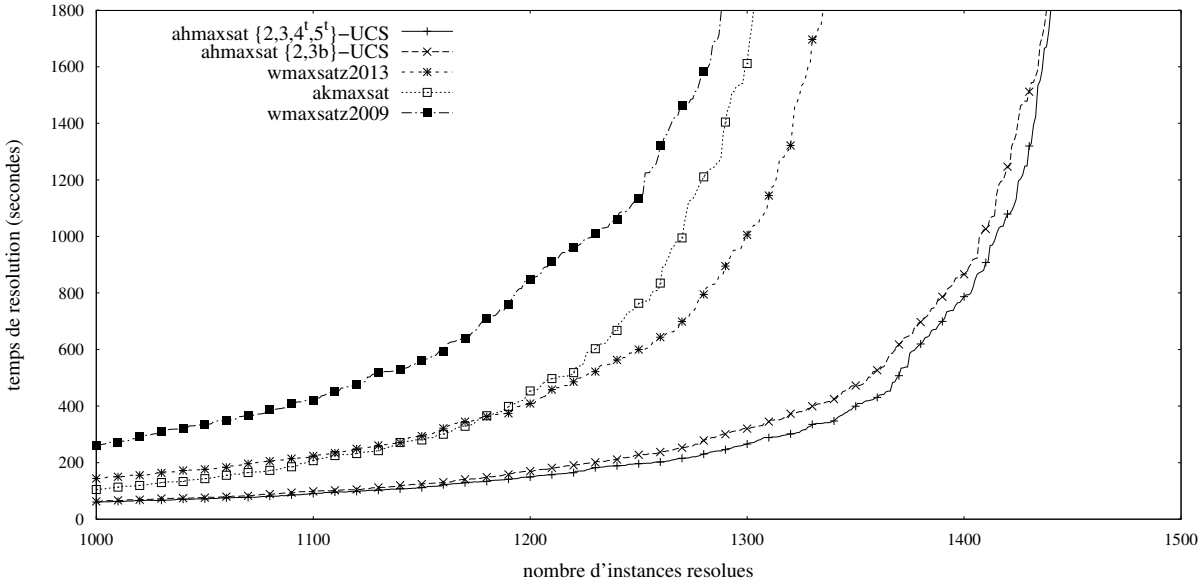


FIGURE 4 – Comparaison des variantes d'AHMAXSAT, avec en abscisse le nombre d'instances résolues et en ordonnée le temps cumulé de résolution.

Même si nous avons décrit ici au travers d'un exemple comment les transformations peuvent réduire le nombre de variables propagées et l'impact que ça a sur la qualité de l'estimation de la BI, les caractéristiques précises des sous-ensembles de clauses concernés par ce phénomène sont méconnues. Une étude approfondie de ces caractéristiques serait d'un grand intérêt dans la perspective d'améliorer les schémas d'apprentissages utilisés par les solveurs S&E.

6 Conclusion

Nous avons présenté dans ce papier de nouveaux ensembles de motifs permettant d'enrichir le schéma d'apprentissage des solveurs de type S&E. L'étude expérimentale que nous avons réalisé montre que l'apprentissage des transformations de certains de ces motifs (les $\{2,3,4^t,5^t\}$-UCS) réduit sensiblement le nombre de nœuds de l'arbre de recherche explorés et le temps de résolution de notre solveur AHMAXSAT.

Les résultats expérimentaux obtenus montrent également que l'apprentissage des transformations des $\{4^b,5^b\}$-UCS réduit la capacité de notre solveur à détecter les sous-ensembles inconsistants, et par là même ses performances. Nous avons décrit ce phénomène, ce qui à notre connaissance n'avait jamais été fait auparavant. Nous étudierons ce phénomène dans le future avec pour objectif d'établir les fondements théoriques de l'apprentissage dans les solveurs S&E pour Max-SAT.

Références

[1] André Abramé and Djamal Habet. Efficient application of max-sat resolution on inconsistent subsets. In Barry O'Sullivan, editor, *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP 2014)*, volume 8656 of *Lecture Notes in Computer Science*, pages 92–107. Springer International Publishing, 2014.

[2] André Abramé and Djamal Habet. Local max-resolution in branch and bound solvers for max-sat. In *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2014)*, pages 336–343. IEEE Computer Society, 2014.

[3] André Abramé and Djamal Habet. Maintaining and handling all unit propagation reasons in exact max-sat solvers. In Stefan Edelkamp and Roman Barták, editors, *Proceedings of the 7th Annual Symposium on Combinatorial Search (SOCS 2014)*, pages 2–9. AAAI Press, 2014.

[4] André Abramé and Djamal Habet. On the extension of learning for max-sat. In Ulle Endriss and João Leite, editors, *Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS 2014)*, volume 241 of *Frontiers in Artificial Intelligence and Applications*, pages 1–10. IOS Press, 2014.

[5] María Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for max-sat. *Artificial Intelligence*, 171(8-9) :606–618, 2007.

[6] Federico Heras and Javier Larrosa. New inference rules for efficient max-sat solving. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*, pages 68–73. AAAI Press, 2006.

[7] Federico Heras, Javier Larrosa, and Albert Oliveras. Minimaxsat : A new weighted max-sat solver. In João Marques-Silva and Karem Sakallah, editors, *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT 2007)*, volume 4501 of *Lecture Notes in Computer Science*, pages 41–55. Springer Berlin / Heidelberg, 2007.

[8] Federico Heras, Javier Larrosa, and Albert Oliveras. Minimaxsat : An efficient weighted max-sat solver. *Journal of Artificial Intelligence Research*, 31 :1–32, 2008.

[9] Adrian Kügel. Improved exact solver for the weighted max-sat problem. In Daniel Le Berre, editor, *Proceedings of the 1st Pragmatics of SAT Workshop (POS 2010)*, volume 8 of *EasyChair Proceedings in Computing*, pages 15–27. EasyChair, 2010.

[10] Javier Larrosa and Federico Heras. Resolution in max-sat and its relation to local consistency in weighted csps. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 193–198. Professional Book Center, 2005.

[11] Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient max-sat solving. *Artificial Intelligence*, 172(2-3) :204–233, 2008.

[12] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Resolution-based lower bounds in maxsat. *Constraints*, 15(4) :456–484, 2010.

[13] Chu Min Li, Felip Manyà, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound max-sat solvers. In Peter van Beek, editor, *Proceedings of the 11th Interantional Conference on Principles and Practice of Constraint Programming (CP 2005)*, volume 3709 of *Lecture Notes in Computer Science*, pages 403–414. Springer Berlin / Heidelberg, 2005.

[14] Chu Min Li, Felip Manyà, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*, pages 86–91. AAAI Press, 2006.

[15] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for max-sat. *Journal of Artificial Intelligence Research*, 30 :321–359, 2007.

[16] João Marques-Silva and Karem A. Sakallah. Grasp : A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5) :506–521, August 1999.

Classes Polynomiales Cachées : de la Théorie à la Pratique *

Achref El Mouelhi **Philippe Jégou** **Cyril Terrioux**

LSIS - UMR CNRS 7296
Aix-Marseille Université
13397 Marseille Cedex 20 (France)
{achref.elmouelhi, philippe.jegou, cyril.terrioux}@lsis.org

Résumé

Dans cette contribution, nous introduisons un cadre nouveau qui a pour objet d'étendre le champ des classes polynomiales. Cette approche est basée sur la transformation de CSP et introduit la notion de *classe polynomiale cachée*. Elle part du constat selon lequel les problèmes réels n'appartiennent pas en général à des classes polynomiales mais peuvent parfois, après simplification (des filtrages par cohérence locale par exemple) en faire partie. Ce papier est un résumé de l'article [2].

Abstract

In this paper, we introduce a new framework that aims to extend the scope of tractable classes. This approach is based on the transformation of CSP and introduced the concept of *Hidden Tractable Class*. It starts from the observation that real problems do not belong in general to tractable classes but can sometimes after simplifications (filterings for example) appear in a tractable class. This paper is a summary of [2].

1 Introduction

La plupart des classes polynomiales sont définies sur la base de propriétés très restrictives qui rendent leur présence au sein des instances réelles quasi inexistante. Elles se retrouvent ainsi très peu exploitées en pratique. Pour contourner cette réalité, nous proposons d'étendre le champ des classes polynomiales de CSP en introduisant le concept de *classes polynomiales cachées*. L'idée est que certaines instances qui n'appartiendraient pas à des classes polynomiales, pourraient, après filtrage par exemple, être transformées en instances figurant dans des classes bien connues.

*Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet TUPLES (ANR-2010-BLAN-0210).

2 Classes polynomiales cachées

Le cadre formel que nous proposons ici est basé sur la notion de *transformation* d'instances. Nous dirons que certaines instances constituent des classes polynomiales « cachées » découvertes par transformation d'instances. À cette fin, nous proposons un cadre général qui peut être utilisé pour de nombreuses classes polynomiales connues et pour l'essentiel des transformation d'instances de CSP, comme les filtrages par cohérence locale par exemple. Comme transformations classiques de CSP, nous trouvons généralement les suppressions de variables, de valeurs, l'ajout de contraintes ou la suppression de tuples dans les relations de compatibilité. Pour définir formellement cette notion, nous rappelons qu'une *instance de CSP* est un triplet (X, D, C) où X est un ensemble de variables, D un ensemble de domaines finis, avec un domaine $D(x)$ pour chaque variable x, et C un ensemble de contraintes. Chaque contrainte c est définie par un couple $(S(c), R(c))$ où $R(c)$ est une *relation de compatibilité* sur les domaines de l'ensemble $S(c) \subseteq X$ qui est la *portée* de la contrainte c.

Définition 1 *Étant donnée une instance de CSP $P = (X, D, C)$, t est dite une* ***transformation*** *de P si $t(P) = (t_{var}(X), t_{dom}(D), t_{cons}(C))$ vérifie :*

- $t_{var}(X) \subseteq X$
- $t_{dom}(D) = \{t_{dom}(D(x')) : x' \in t_{var}(X)$ *et* $t_{dom}(D(x')) \subseteq D(x)\}$
- $\forall c \in C, t_{cons}(c)$ *vérifie :*
  - $t_{cons}(S(c)) = S(c) \backslash \{x \in X : x \notin t_{var}(X)\}$ *et*
  - $t_{cons}(R(c)) \subseteq R(c)[t_{cons}(S(c))]$.
- $\forall c' \in t_{cons}(C)$ *telle que* $c' \notin C$, $t_{cons}(R(c')) \subseteq \prod_{x' \in S(c')} t_{dom}(D(x'))$.

D'autres transformations sont envisageables, mais nous nous limitons à celles-ci dans le cadre de ce travail. Cette définition permet d'introduire la notion de classe cachée :

Définition 2 *Étant données une transformation t et une classe d'instances $\mathcal{P}$, $t(\mathcal{P}) = \{t(P) : P \in \mathcal{P}\}$. Étant donnée une classe $\mathcal{C}$, la classe d'instances* ***mise en évidence par*** *t* ***pour*** *$\mathcal{C}$ est $\mathcal{C}^t = \{P : t(P) \in \mathcal{C}\}$. La classe $\mathcal{P}$ est dite* ***classe cachée*** *de $\mathcal{C}$ pour t, si $\mathcal{P} \subseteq \mathcal{C}^t$ i.e. si $t(\mathcal{P}) \subseteq \mathcal{C}$. Enfin, $\mathcal{P}$ est appelée* ***classe polynomiale cachée de*** *$\mathcal{C}$* ***pour*** *t si $\mathcal{C}$ est une classe polynomiale et si t préserve la cohérence et peut être calculée en temps polynomial.*

Ainsi, si $\mathcal{C}$ est une classe polynomiale et si $\mathcal{P}$ est une classe cachée de $\mathcal{C}$ pour une transformation t calculable en temps polynomial, pour le cas d'une instance $P \in \mathcal{P}$, il sera suffisant d'appliquer t sur P pour avoir une version modifiée de P qui sera traitable en temps polynomial. Au-delà, $\mathcal{P}$ sera aussi considérée comme classe polynomiale. Notre démarche ayant pour objet d'étendre des classes polynomiales à l'aide de transformations d'instances, il est naturel d'étudier la puissance relative de ces transformations. Nous formalisons cela par des comparaisons entre classes mises en évidence par des transformations t pour des classes $\mathcal{C}$.

Définition 3 *Étant données deux transformations t_1 et t_2, et deux classes $\mathcal{C}_1$ et $\mathcal{C}_2$, on dit que $\mathcal{C}_2^{t_2}$ est* ***plus grande*** *que $\mathcal{C}_1^{t_1}$ si $\mathcal{C}_1^{t_1} \subseteq \mathcal{C}_2^{t_2}$. De plus, on dit que $\mathcal{C}_2^{t_2}$ est* ***strictement plus grande*** *que $\mathcal{C}_1^{t_1}$ si $\mathcal{C}_1^{t_1} \subsetneq \mathcal{C}_2^{t_2}$, et on dit que $\mathcal{C}_1^{t_1}$ et $\mathcal{C}_2^{t_2}$ sont* ***incomparables*** *si aucune relation entre elles n'existe. Finalement, on dit que $\mathcal{C}_1^{t_1}$ et $\mathcal{C}_2^{t_2}$ sont* ***égales*** *si $\mathcal{C}_1^{t_1} = \mathcal{C}_2^{t_2}$.*

Sur la base de ces relations, il est possible de définir un certain nombre de propriétés qui sont détaillées dans [2], offrant ainsi un cadre d'étude général pour la comparaison de classes et de transformations. Par exemple, on peut facilement constater que pour toute paire de classes $\mathcal{C}_1$ et $\mathcal{C}_2$ telles que $\mathcal{C}_1 \subseteq \mathcal{C}_2$, et pour toute transformation t, on a $\mathcal{C}_1^t \subseteq \mathcal{C}_2^t$. Cette contribution ayant pour objet d'étendre l'intérêt pratique des classes polynomiales, il est naturel d'étudier expérimentalement l'étendue des classes cachées.

3 Le cas de BTP et des filtrages

L'étude de la classe polynomiale BTP [1] semble assez naturelle ici car elle constitue l'une des classes les plus larges qui aient été proposées jusqu'à présent. De même, coupler son étude avec les techniques de filtrage semble tout à fait naturel car ce sont les transformations les plus exploitées en pratique, que ce soit en prétraitement ou bien alors pendant une recherche par des algorithmes tels que MAC par exemple. Aussi, nous avons étudié les classes cachées de BTP qui peuvent être découvertes par les filtrages usuels tels que AC, SAC, PIC, $maxRPC$, RPC, PC, SPC ou encore NIC. Nous avons pour cela considéré les instances de la compétition CSP 2008, et dénombré celles qui apparaissent dans des classes cachées. Alors qu'à la base, très peu d'instances figurent dans BTP (précisément 12), la table 1 permet d'observer qu'elles sont finalement plus nombreuses dès lors qu'elles sont recherchées au sein de classes cachées. Ces résultats s'avèrent en adéquation avec les comparaisons théoriques présentées dans la figure 1 où un arc entre deux classes exprime la relation *strictement plus grande* et les pointillés l'*incomparabilité*.

	BTP	BTP^{AC}	BTP^{PIC}	BTP^{maxRPC}	BTP^{SAC}	BTP^{NIC}	BTP^{SPC}
# inst.	12	191	400	493	550	900	594
# cons.	-	46	47	47	47	83	71

TABLE 1 – Instances et classes cachées de BTP.

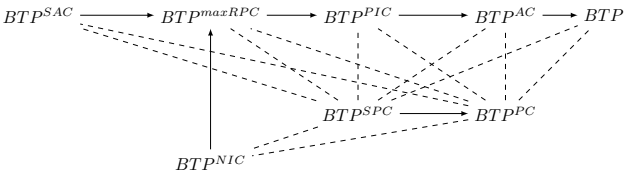


FIGURE 1 – Relations entre classes cachées de BTP.

4 Conclusion

Souvent ignorée des praticiens pour de bonnes raisons, l'étude des classes polynomiales s'est cantonnée à des travaux théoriques. La notion de classe cachée introduite ici pourrait lui donner un impact plus large au sein de la communauté de la Programmation par Contraintes. D'une part, elle permettrait d'offrir une explication à l'efficacité parfois surprenante des solveurs généraux qui se trouve en contradiction avec le verdict délivré par l'analyse de la complexité dans le pire des cas. D'autre part, elle pourrait orienter les recherches théoriques en les dirigeant vers l'élaboration de classes polynomiales plus en adéquation avec les algorithmes de résolution souvent à base de filtrages.

Références

[1] M. Cooper, P. Jeavons, and A. Salamon. Generalizing constraint satisfaction on trees : hybrid tractability and variable elimination. *Artificial Intelligence*, 174 :570–584, 2010.

[2] Achref El Mouelhi, Philippe Jégou, and Cyril Terrioux. Hidden tractable classes : From theory to practice. In *Proceedings of the 26th IEEE ICTAI*, pages 437–445, 2014.

Une contrainte de No-Overlap dépendant du temps : Application aux problèmes de tournées de livraison en milieu urbain

Penélope Aguiar Melgarejo[1,2] **Philippe Laborie**[2] **Christine Solnon**[1]

[1] LIRIS, INSA Lyon, Lyon, France
[2] France Lab, IBM, Gentilly, France
{penelopeam,laborie}@fr.ibm.com christine.solnon@insa-lyon.fr

Résumé

Nous nous intéressons au problème du voyageur de commerce dépendant du temps (*Time-Dependent Traveling Salesman Problem* / TDTSP), une version étendue du voyageur de commerce où la durée du trajet entre deux livraisons varie en fonction de l'heure du voyage. Dans [1] nous proposons un nouveau benchmark pour le TDTSP et nous montrons l'intérêt de prendre en compte le fait que les durées varient. Nous présentons une nouvelle contrainte globale (étendant No-Overlap) qui intègre des durées variant en fonction du temps et nous montrons que cette nouvelle contrainte est plus performante que le modèle classique de programmation par contraintes.

Abstract

The Time-Dependent Traveling Salesman Problem (TDTSP) is the extended version of the TSP where arc costs depend on the time when the arc is traveled. When we consider urban deliveries, travel times vary considerably during the day and optimizing a delivery tour comes down to solving an instance of the TDTSP. In [1] we propose a benchmark for the TDTSP based on real traffic data and show the interest of handling time dependency in the problem. We then present a new global constraint (an extension of No-Overlap) that integrates time-dependent transition times and show that this new constraint outperforms the classical CP approach.

1 Introduction

Quand nous considérons les problèmes réels d'optimisation, le temps est généralement une dimension importante à prendre en compte. Cela est particulièrement le cas pour les problèmes de livraison pour lesquels le temps est fréquemment présent sous différentes formes : temps de déplacement entre deux livraisons consécutives ; fenêtres de temps autorisées pour chaque livraison ; contraintes de précédence entre livraisons. Une source supplémentaire de complexité de ces problèmes apparaît lorsque le temps de parcours entre points de livraison dépend de l'heure de départ, typiquement parce que dans les zones urbaines les conditions de circulation varient beaucoup au cours de la journée. Nous nous intéressons plus particulièrement au problème du voyageur de commerce dépendant du temps (*Time-Dependent Traveling Salesman Problem* / TDTSP), une version étendue du voyageur de commerce (TSP) où le temps de trajet entre deux livraisons varie en fonction de l'heure du voyage.

Puisque la disponibilité de données réelles dans ce domaine est assez récente, ces problèmes n'ont pas été très étudiés dans la littérature et les approches utilisant la Programmation par Contraintes (PPC) sont encore plus rares. D'une part, la PPC est généralement moins efficace que la programmation linéaire en nombres entiers ou les méta-heuristiques pour les problèmes de tournées de véhicules purs (qui ne dépendent pas du temps). D'autre part, le "Constraint-Based Scheduling" [2], qui est l'application de la PPC aux problèmes d'ordonnancement, est l'un des plus grand succès industriels de la PPC et a montré que les technologies de PPC peuvent être très efficaces pour résoudre les problèmes temporels. Des nouveaux types de variables (d'intervalle) et de contraintes globales associées ainsi que des nouveaux algorithmes de recherche ont été développés récemment [4, 5, 6] pour améliorer l'expressivité et l'efficacité des modèles PPC impliquant les domaines temporels.

Nous avons introduit un nouveau benchmark [1] pour le TDTSP généré à partir des données réelles de traffic, provenant de la ville de Lyon. Nous présentons aussi une nouvelle contrainte globale (TDNoOverlap) créée pour exploiter de façon plus efficace les données dépendant du temps. Cette contrainte a été implémentée sur le moteur CP Optimizer de IBM ILOG CPLEX Optimization Studio et testé expérimentalement sur notre nouveau benchmark.

2 La contrainte `TDNoOverlap`

`TDNoOverlap` prend comme paramètres :
- une fonction T donnant pour chaque couple de sommets (i, j) et chaque instant t la durée pour aller de i à j quand on part de i à l'instant t ;
- une séquence de variables correspondant aux heures d'arrivée et départ sur chaque sommet ;

et impose que les heures de départ et d'arrivée entre deux sommets consécutifs respectent les durées de transition données par T. La contrainte crée un graphe de précédence où les arcs correspondent aux relations de précédence entre les visites de sommets. Deux types d'arcs sont considérés entre deux visites i et j :

1. Un arc **next** (successeur immédiat) $i \Rightarrow j$ implique que j sera visité juste après i.
2. Un arc **successor** $i \rightarrow j$ implique que j sera visité après i mais d'autres visites peuvent être faites entre les deux.

Pendant la recherche, des nouveaux arcs **next** et **successor** sont rajoutés au graphe de précédence soit à cause de décisions prises soit à cause des propagations. Le graphe de précédence maintient incrémentalement la fermeture transitive des arcs. Dans [1] nous montrons les avantages de considérer ces deux types d'arcs plutôt que juste les arcs de type **next** pour les problèmes asymétriques comme le TDTSP et nous décrivons en détails les différentes propagations temporelles qui sont faites pour chaque type d'arc.

3 Résultats expérimentaux

Nous étendons le modèle PPC classique pour le TSP [3] pour prendre en compte les durées de transition qui dépendent du temps et nous le comparons à un modèle qui utilise la contrainte `TDNoOverlap`. Dans un premier temps nous comparons les capacités de filtrage des deux modèles en utilisant la même stratégie de recherche pour les deux modèles : ordonnancement chronologique des visites et choix de la visite la plus proche en termes de temps de transition d'abord, la date au plus tôt de la visite précédente étant connue. Nous avons mesuré le nombre de branches et le temps CPU des deux approches sur les 180 instances de taille 10 de notre benchmark [2]. Le modèle avec `TDNoOverlap` propage beaucoup plus que le modèle PPC classique (de l'ordre de 50 fois moins de branches) et trouve aussi de meilleures solutions plus vite. Pour les instances de taille 10, la recherche se finit 100 plus vite en moyenne.

Nous avons d'autre part comparé les coûts des meilleures solutions trouvées pour les instances de taille 20 et 30 par les deux approches en utilisant la recherche automatique de CP Optimizer (plus performante qu'une simple recherche en profondeur). La même heuristique de recherche qu'avant a été utilisée avec une limite de temps de 900s. Pour les instances de taille 20, le modèle avec `TDNoOverlap` trouve et prouve la solution optimale pour 165 instances sur les 180 et, en moyenne, la solution trouvée est au moins 10% meilleure que celle trouvée par le modèle classique de PPC. Pour les instances à 30 visites, les deux modèles sont incapables de prouver l'optimalité dans la limite fixée mais, en moyenne, les solutions trouvées par `TDNoOverlap` sont au moins 20% meilleures que celles trouvées par le modèle PPC classique.

1. Disponible sur liris.cnrs.fr/christine.solnon/TDTSP.html

2. Comparaison effectuée seulement sur les instances de taille 10 puisque le modèle classique de PPC n'est capable de résoudre les problèmes plus grands à l'optimalité.

Références

[1] P. Aguiar Melgarejo, P. Laborie, and C. Solnon. A time-dependent no-overlap constraint : Application to urban delivery problems. In *Proceedings of the 12th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*, 2015.

[2] P. Baptiste, P. Laborie, C. Le Pape, and W. Nuijten. Constraint-based scheduling and planning. In *Handbook of Constraint Programming*. Elsevier, 2006.

[3] Pascal Benchimol, Willem Jan van Hoeve, Jean-Charles Regin, Louis-Martin Rousseau, and Michel Rueher. Improved filtering for weighted circuit constraints. *Constraints*, pages 205–233, 2012.

[4] P. Laborie and D. Godard. Self-adapting large neighborhood search : Application to single-mode scheduling problems. In *MISTA*, 2007.

[5] P. Laborie and J. Rogerie. Reasoning with conditional time-intervals. In *FLAIRS Conference*, 2008.

[6] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. Reasoning with conditional time-intervals. part ii : An algebraical model for resources. In *FLAIRS Conference*, 2009.

RLBS: Une stratégie de retour arrière adaptative basée sur l'apprentissage par renforcement pour l'optimisation combinatoire

Ilyess Bachiri[1,2] **Jonathan Gaudreault**[1,2] **Brahim Chaib-draa**[2] **Claude-Guy Quimper**[1,2]

[1] FORAC Research Consortium, Québec, Canada
[2] Université Laval, Québec, Canada
ilyess.bachiri@cirrelt.ca jonathan.gaudreault@forac.ulaval.ca
{brahim.chaib-draa,claude-guy.quimper}@ift.ulaval.ca

Résumé

Les problèmes d'optimisation combinatoire sont souvent très difficiles à résoudre et le choix d'une stratégie de recherche joue un rôle crucial dans la performance du solveur. Une stratégie de recherche est dite *adaptative* quand elle s'adapte dynamiquement à la structure du problème et est capable d'identifier les zones de l'espace de recherche qui contiennent les bonnes solutions. Nous proposons un algorithme (RLBS) qui apprend à faire des retours arrière de manière efficace lors de l'exploration d'arbres non-binaires. Le branchement est effectué en utilisant une stratégie de choix de variable/valeur quelconque. Cependant, quand un retour arrière est nécessaire, la sélection du nœud cible s'appuie sur l'apprentissage automatique. Comme les arbres sont non-binaires, on a l'occasion de faire des retours arrière à plusieurs reprises vers chaque nœud durant l'exploration, ce qui permet d'apprendre quels nœuds mènent vers les meilleures récompenses en général (c'est-à-dire, vers les feuilles les plus intéressantes). RLBS est évalué sur un problème de planification en utilisant des données industrielles. Il surpasse aussi bien des stratégies d'exploration classiques (non-adaptative) (DFS, LDS) qu'une stratégie adaptative (IBS).

1 Introduction

Les problèmes d'optimisation combinatoire (Constraint Optimization Problem – COP) sont souvent très difficiles à résoudre et le choix de la stratégie de recherche a une influence importante sur la performance du solveur. Afin de résoudre un problème d'optimisation combinatoire en explorant un arbre de recherche, il faut choisir une heuristique de choix de variable (qui définie l'ordre dans lequel les variables vont être instanciées), une heuristique de choix de valeurs (qui définie l'ordre dans lequel les valeurs seront essayées), et une stratégie de retour arrière (qui détermine vers quel nœud effectuer les retours arrière lorsqu'une feuille de l'arbre est rencontrée). Quelques politiques de retours arrière sont totalement déterministes (e.g. Depth-First Search, DFS) pendant que d'autres s'appuient sur des mécanismes d'évaluation de nœuds plus dynamiques (e.g. Best-First Search). Certaines (e.g. Limited Discrepancy Search [9]) peuvent être implémentées soit comme un algorithme itératif déterministe ou un évaluateur de nœud [3].

Une stratégie est dite *adaptative* quand elle s'adapte dynamiquement à la structure du problème et identifie les zones de l'espace de recherche qui contiennent les bonnes solutions. Des stratégies de branchement adaptatives ont été proposées (e.g. Impact-Based Search (IBS) [17]) ainsi qu'une stratégie de retour arrière adaptative (e.g. Adaptive Discrepancy Search [7], proposée pour les problèmes d'optimisation distribués).

Dans cet article, nous proposons une stratégie de retour arrière qui se base sur l'apprentissage automatique pour améliorer la performance du solveur. Plus particulièrement, nous utilisons l'apprentissage par renforcement pour identifier les zones de l'espace de recherche qui contiennent les bonnes solutions. Cette approche a été développée pour les problèmes d'optimisation combinatoire dont l'espace de recherche est encodé dans un arbre non-binaire. Comme les arbres sont non-binaires, on a l'occasion d'effectuer plusieurs retours arrière vers chaque nœud durant l'exploration. Ceci permet d'apprendre quels nœuds mènent vers les

meilleures récompenses en général (c'est-à-dire, vers les feuilles les plus intéressantes).

Le reste de cet article est organisé comme suit : La section 2 passe en revue quelques concepts préliminaires relatifs à la recherche adaptative et l'apprentissage par renforcement. La section 3 explique comment le retour arrière peut être formulé en tant que tâche d'apprentissage par renforcement et introduit l'algorithme proposé (*Reinforcement Learning Backtracking Search*, or RLBS). La section 4 présente les résultats pour un problème industriel complexe qui combine planification et ordonnancement. RLBS est comparé à des stratégies de recherche classiques (non-adaptatives) (DFS, LDS) ainsi qu'à une stratégie de branchement adaptative (IBS). La section 5 conclue l'article.

2 Concepts préliminaires

La résolution des problèmes d'optimisation combinatoire en utilisant l'exploration globale se réduit à définir trois éléments-clés : une stratégie de choix de variable, une stratégie de choix de valeur, et une stratégie de retour-arrière [20]. L'espace de recherche est structuré sous forme d'un arbre dont la topologie est définie par les stratégies de choix de variable/valeur. Peu importe les stratégies de choix de variable/valeur employées, l'arbre de recherche couvre entièrement l'espace de recherche. La stratégie de retour-arrière détermine l'ordre dans lequel les nœuds vont être visités. Les stratégies de retour-arrière peuvent être implémentées sous forme d'algorithmes itératifs, ou de mécanismes d'évaluation de nœud [3].

2.1 Heuristiques de choix de variable/valeur et apprentissage

Quelques algorithmes apprennent durant l'exploration quelles variables sont les plus difficiles à instancier, dans le but de changer l'ordre d'instanciation des variables dynamiquement (e.g. YIELDS [10]). Dans [4] et [8], à chaque fois qu'une contrainte entraîne un échec, la priorité des variables impliquées dans cette contrainte est augmentée.

Dans Impact Based Search (IBS) [17], l'impact des variables est mesuré en observant à quel point leurs instanciations réduit la taille de l'espace de recherche. Comme IBS choisit en même temps la variable à instancier et la valeur à lui attribuer, on peut dire qu'il apprend une combinaison de stratégie de choix de variable et de stratégie de choix de valeur.

2.2 Apprendre à effectuer les retours arrière

Les approches où le système apprend à évaluer la qualité des nœuds sont d'un grand intérêt quand il s'agit de stratégies de retour arrière. Ruml [18] propose une approche intéressante à cet égard. Pendant que LDS basique attribue le même degré d'importance à toutes les déviations, Best Leaf First Search (BLFS) donne différents poids aux déviations selon leur profondeur. BLFS utilise une régression linéaire afin de déterminer la valeur des poids. Le modèle n'a pas vraiment été utilisé pour définir une stratégie de retour arrière. Au lieu de ceci, l'algorithme d'exploration effectue des descentes successives dans l'arbre. L'algorithme de Ruml a abouti à de très bons résultats (voir [19]), et a été l'inspiration derrière le prochain algorithme.

Adaptive Discrepancy Search (ADS) [7] est un algorithme qui a été proposé pour l'optimisation distribuée mais qui peut être employé dans un contexte de COP classique. Durant la recherche, il apprend dynamiquement vers quels nœuds il est le plus profitable d'effectuer des retours arrière (afin de concentrer les efforts sur ces zones de l'arbre en premier). Pour chaque nœud, il tente d'apprendre une fonction $Improvement(i)$ qui prédit à quel point la première feuille rencontrée après avoir effectué un retour arrière vers ce nœud pour la i-ème fois sera de bonne qualité, en comparaison avec les retours arrière précédents effectués vers le même nœud. L'inconvénient de cette méthode est qu'une fonction doit être apprise pour chacun des nœuds ouverts. Ces fonctions doivent aussi être mises à jour à chaque fois qu'un nœud mène vers une nouvelle solution [13] (même si une approximation peut être calculée en utilisant la régression).

2.3 Apprentissage par renforcement

L'idée fondamentale derrière l'apprentissage par renforcement est de trouver une façon d'associer les actions aux situations afin de maximiser la récompense totale. L'apprenti ne sait pas quelles actions prendre, il doit découvrir quelles actions mènent vers les plus grandes récompenses (à long terme). Les actions peuvent affecter non seulement la récompense immédiate mais aussi la situation prochaine et, au final, toutes les récompenses qui suivent [2]. De plus, les actions peuvent ne pas mener vers les résultats espérés à cause de l'incertitude de l'environnement.

L'apprentissage par renforcement s'appuie sur un cadre formel qui définit l'interaction entre l'apprenti et l'environnement en termes d'états, d'actions, et de récompenses. L'environnement qui supporte l'apprentissage par renforcement est typiquement formulé en tant qu'un processus décisionnel de Markov (Markov

Decision Process - MDP) à états finis. Dans chaque état $s \in S$, un ensemble d'actions $a \in A$ sont disponibles, parmi lesquelles l'apprenti doit choisir celle qui maximise la récompense cumulative. L'évaluation des actions est entièrement basée sur l'expérience de l'apprenti, cumulée à travers ses interactions avec l'environnement. Le but de l'apprenti est de trouver une politique optimale $\pi : S \to A$ qui maximise la récompense cumulative. La récompense cumulative est soit définie en tant que la somme de toutes les récompenses $R = r_0 + r_1 + \cdots + r_n$, soit exprimée en tant que la somme des récompenses dévaluées $R = \sum_t \gamma^t r_t$. Le facteur de dévaluation $0 \leq \gamma \leq 1$ est appliqué pour privilégier les récompenses récentes. La représentation sous forme de somme dévaluée de la récompense est utilisée pour un MDP sans état final.

L'intuition centrale derrière l'apprentissage par renforcement est que les actions qui mènent vers les grandes récompenses doivent être choisie plus souvent.

Dans une tâche d'apprentissage par renforcement, chaque action a, dans chaque état s, est associé à une valeur numérique $Q(s, a)$ qui représente la désirabilité de choisir l'action a dans l'état s. Ces valeurs sont appelées Q-Values. Plus grande est la Q-Value, plus c'est probable que l'action associée mène vers une bonne solution, selon le jugement de l'apprenti. À chaque fois qu'une récompense est retournée à l'apprenti, il doit mettre à jour la Q-Value de l'action qui a mené à cette récompense. Cependant, l'ancienne Q-Value ne devrait pas être complètement oubliée, sinon l'apprenti se baserait uniquement sur la toute dernière expérience à chaque fois qu'il doit prendre une décision. Pour ce faire, on garde une partie de l'ancienne Q-Value et on la met à jour avec une partie de la nouvelle expérience. Aussi, on suppose que l'apprenti va agir de manière optimale par la suite. De plus, les récompense futures espérées doivent être dévaluées pour privilégier les récompenses les plus récentes.

Soit s l'état courant, s' l'état prochain, a une action, r la récompense retournée après avoir pris l'action a, α le taux d'apprentissage, et γ le facteur de dévaluation. La formule de mise à jour des Q-Values est la suivante :

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')] \quad (1)$$

2.4 Apprentissage automatique et exploration

L'idée d'utiliser l'apprentissage par renforcement pour résoudre les problèmes d'optimisation combinatoires est exploitée par différents travaux de recherche [14, 16, 21]. Des chercheurs ont essayé d'appliquer l'apprentissage par renforcement pour résoudre des problèmes d'optimisation alors que d'autres l'ont employé dans le contexte de problèmes de satisfaction de contraintes (Constraint Satisfaction Problems – CSP).

À titre d'exemple, Xu et al. [21] ont proposé une formulation de la résolution du CSP en tant que tâche d'apprentissage par renforcement. Un ensemble d'heuristiques de choix de variable est fourni à l'algorithme qui apprend laquelle utiliser, et quand l'utiliser, afin de résoudre un CSP dans le plus bref délai. Le processus d'apprentissage prend place de manière *offline* et est mené sur différentes instances du même CSP. Les états sont les instances ou sous-instances du CSP. Les différentes heuristiques de choix de variable définissent les actions. Une récompense est retournée à chaque fois qu'une instance est résolue. Cette approche se base sur le Q-Learning pour apprendre l'heuristique de choix de variable optimale à chaque point de décision dans l'arbre de recherche, pour une (sous)-instance du CSP. En d'autres termes, le processus d'apprentissage détermine, dans chaque contexte, quelle heuristique de choix de variable est le plus efficace.

Par ailleurs, Loth et al. [11] ont proposé l'algorithme BASCOP (*Bandit Search for Constraint Programming*) qui guide l'exploration dans le voisinage de la dernière meilleure solution trouvée. Cet algorithme se base sur des estimations calculées à travers plusieurs redémarrage. BASCOP a été testé sur un problème de *job shop* [12] et a donné des résultats similaires à ce qui existait dans l'état de l'art de la programmation par contrainte.

Une technique de recherche locale qui utilise l'apprentissage par renforcement est aussi proposée dans [14]. Cette approche consiste en la résolution des COPs en se basant sur une population d'agents d'apprentissage par renforcement. Les paires $\langle$variable, valeur$\rangle$ sont considérés comme les états de la tâche d'apprentissage par renforcement, et les stratégies de branchement définissent les actions. Chaque agent se voit assigné une zone spécifique de l'espace de recherche où il est censé apprendre et trouver de bonnes solutions locales. L'expertise de toute la population d'agents est ensuite exploitée. Une nouvelle solution globale est produite en gardant une partie de la meilleure solution locale trouvée par un agent, et complétée en utilisant l'expertise d'un autre agent de la population.

Selon [16], la recherche locale peut être formulée comme une politique de résolution dans un processus décisionnel de Markov (MDP) où les états représentent les solutions, et les actions définissent les solutions voisines. Les techniques d'apprentissage par renforcement peuvent, dès lors, être utilisées pour apprendre une fonction de coût afin d'améliorer la recherche locale. Une manière de faire serait d'apprendre une nouvelle fonction de coût sur de multiples trajectoires de

la même instance. L'algorithme STAGE de Boyan et Moore [5] procède ainsi et alterne entre l'utilisation de la fonction de coût originale et la fonction de coût nouvellement apprise. En améliorant la précision de prédiction de la fonction de coût apprise, la capacité des heuristiques à guider la recherche s'améliore.

Une autre approche qui utilise l'apprentissage par renforcement pour améliorer la recherche locale dans le contexte de l'optimisation combinatoire est d'apprendre une fonction de coût de manière *offline*, et ensuite l'utiliser sur de nouvelles instances du même problème. Les travaux de Zhang et Dietterich [22] se situent dans cette catégorie.

3 RLBS : Le retour arrière en tant que tâche d'apprentissage par renforcement

Cette section introduit *Reinforcement Learning Backtracking Search* (RLBS). Le branchement est effectué selon n'importe quelle heuristique de choix de variable/valeur. À chaque fois qu'une feuille/solution est atteinte, un nœud vers lequel effectuer un retour arrière doit être sélectionné. À chaque candidat au retour arrière (c'est-à-dire un nœud avec au moins un enfant non-visité) correspond une *action* ("effectuer un retour arrière vers ce nœud"). Une fois un nœud est sélectionné, l'exploration de l'arbre de recherche poursuit à partir de ce nœud jusqu'à ce qu'une feuille/solution est rencontrée. La différence entre la qualité de cette nouvelle solution et la meilleure solution trouvée jusqu'à date constitue la *récompense* retournée pour avoir choisi l'action précédente. Comme les arbres de recherche sont non-binaires, on est amené à effectuer plusieurs retours arrière vers chaque nœud durant l'exploration. Ceci permet d'identifier les actions qui payent le plus (à savoir, les nœuds qui ont le plus de chance de mener vers des feuilles/solutions intéressantes).

Cette situation est similaire au problème du *k-armed bandit* [1]. Il s'agit d'un problème d'apprentissage par renforcement mono-état. Plusieurs actions sont possibles (tirer sur l'un des bras de la machine à sous). Chaque action peut mener vers une récompense (de manière stochastique) et un équilibre doit être maintenu entre l'exploration et l'exploitation. Dans notre situation de retour arrière, choisir une action mène à la découverte de nouveaux nœuds (nouvelles actions), en plus de retourner une récompense (ce qui est stochastique et non-stationnaire).

3.1 Apprentissage

Comme dans le contexte d'apprentissage par renforcement classique, l'évaluation (Q-value) d'une action a est mise à jour à chaque fois qu'une récompense est retournée après avoir choisi une action. Comme il s'agit d'un environnement mono-état, le facteur de dévaluation γ est égal à 0 et l'équation (1) se réduit à l'équation (2) :

$$Q(a) \leftarrow Q(a) + \alpha[r(a) - Q(a)] \qquad (2)$$

où $r(a)$ est la récompense et α est le taux d'apprentissage.

La prochaine action à effectuer est sélectionnée en se basant sur ces évaluations. Un nœud qui a généré une grande récompense au début mais qui n'a jamais mené vers de bonnes solutions par la suite verra sa Q-Value décroître au fil du temps, jusqu'à ce qu'il devienne moins intéressant que les autres nœuds/actions.

3.2 Initialisation de l'algorithme

Au début de l'exploration, on descend jusqu'à la première feuille/solution de l'arbre en utilisant un DFS. Puis, on effectue un retour arrière une fois vers chacun des nœuds ouverts (ceci est similaire à réaliser les 2 premières itérations de LDS), ce qui permet de calculer leurs Q-Values. Ensuite, on commence à se baser sur les Q-Values pour choisir le prochain nœud pour effectuer le retour arrière. Quand un nœud est visité pour la première fois, sa Q-Value est initialisée par celle de son parent.

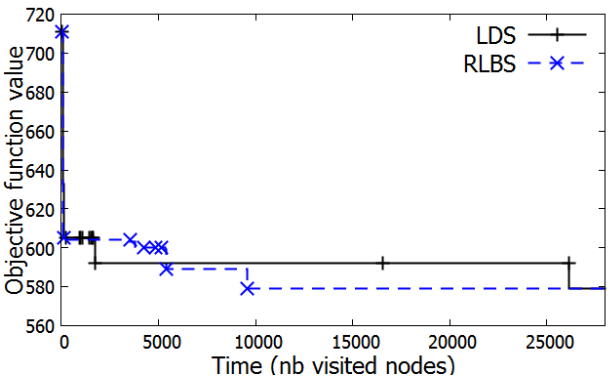


FIGURE 5 – La valeur de la fonction objectif en fonction du temps de calcul de LDS et RLBS pour le cas #5

4 Expérimentations en utilisant des données industrielles

L'objectif principal de ces travaux de recherche était de trouver une façon plus intelligente de sélectionner le nœud vers lequel effectuer le retour-arrière durant l'exploration globale d'un arbre de recherche. Et ce pour (1) des problèmes d'optimisation combinatoires

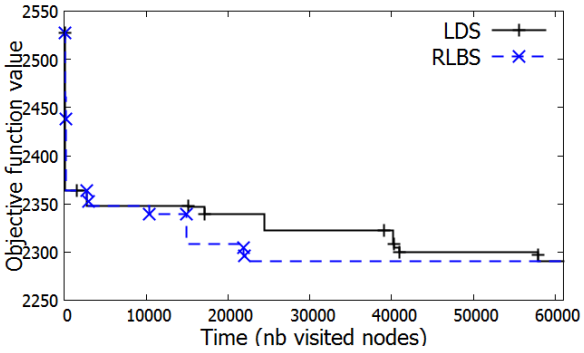


FIGURE 1 – La valeur de la fonction objectif en fonction du temps de calcul de LDS et RLBS pour le cas #1

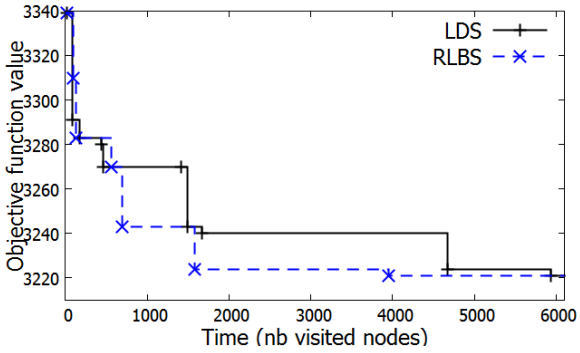


FIGURE 2 – La valeur de la fonction objectif en fonction du temps de calcul de LDS et RLBS pour le cas #2

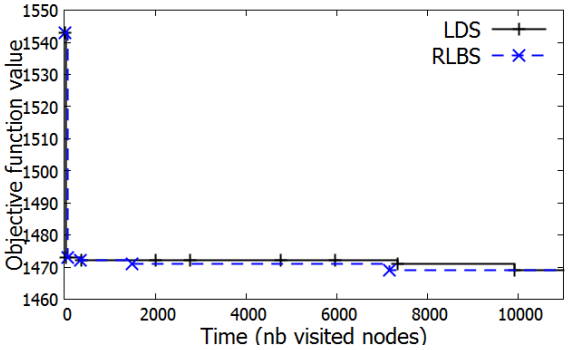


FIGURE 3 – La valeur de la fonction objectif en fonction du temps de calcul de LDS et RLBS pour le cas #3

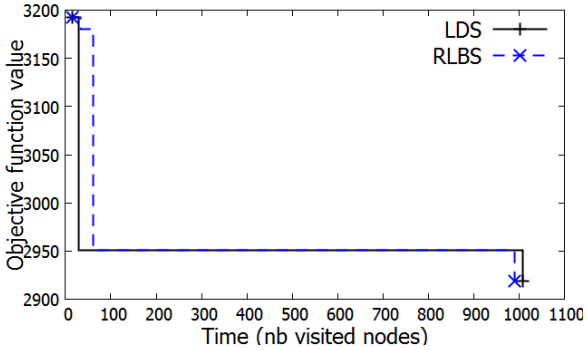


FIGURE 4 – La valeur de la fonction objectif en fonction du temps de calcul de LDS et RLBS pour le cas #4

(2) pour lesquels on connaît déjà de bonnes stratégies de choix de variable/valeur. Des expérimentations ont été menées sur un problème de planification et ordonnancement issu de l'industrie du bois (le problème de planification et ordonnancement des opérations de rabotage du bois) car il répond parfaitement aux critères mentionnés précédemment.

Ce problème est difficile car il traite des processus *divergents* avec de la *coproduction* : un processus génère plusieurs produits différents en même temps, à partir d'un même produit de matière première. De plus, plusieurs processus alternatifs peuvent produire le même produit. Finalement, il implique des règles de mis en course complexes. Le but est de minimiser les retards de livraison.

Le problème est décrit plus en détails dans [6] qui fournit de bonnes heuristiques de choix de variable/valeur spécifiques à ce problème. Dans [7], ces heuristiques ont été employées pour guider la recherche dans un modèle de programmation par contraintes. Muni de ces heuristiques, LDS surpasse DFS ainsi qu'une autre approche de programmation mathématique. Dans [15], de la parallélisation a été utilisée pour améliorer la performance. Cela dit, l'ordre de visite des nœuds est le même que pour la version centralisée, donc elle implémente la même stratégie.

Les mêmes heuristiques de choix de variable/valeur du travail précédent ont été employées. Aussi, les mêmes données industrielles fournies par une compagnie canadienne de l'industrie du bois ont été utilisées. Cependant, afin de comparer les algorithmes en termes de temps de calcul nécessaire pour trouver les solutions optimales, on a dû réduire la taille des problèmes (5 périodes au lieu de 44).

RLBS a été testé avec un taux d'apprentissage $\alpha = 0.5$. Il a été comparé à LDS (qui privilégie les nœuds avec le moins de déviations).

Les figures 1 à 5 présentent les résultats pour cinq cas différents. Le tableau 1 montre la réduction du temps de calcul (mesuré en tant que le nombre de nœuds visités) nécessaire pour obtenir une solution optimale. RLBS réduit le temps de calcul dans chacun des cinq cas (de 37.66% en moyenne).

Comme dans le contexte industriel on n'a souvent

TABLE 1 – Temps de calcul nécessaire pour trouver la meilleure solution (RLBS vs. LDS)

	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**	**Average**
LDS	57926	5940	9922	1008	26166	20192.4
RLBS	22164	3949	7172	990	9545	8764
Time reduction	↓ 61.73%	↓ 33.52%	↓ 27.72%	↓ 1.79%	↓ 63.52%	↓ 37.66%

TABLE 2 – Moyenne de temps de calcul pour trouver une solution d'une qualité quelconque (RLBS vs. LDS)

	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**	**Average**
LDS	8777.82	1243.86	386.5	127.34	2776.05	2662.31
RLBS	4254.81	606.79	264.24	152.17	1320.15	1319.63
Time reduction	↓ 51.53%	↓ 51.22%	↓ 31.63%	↑ 19.5%	↓ 52.44%	↓ 33.46%

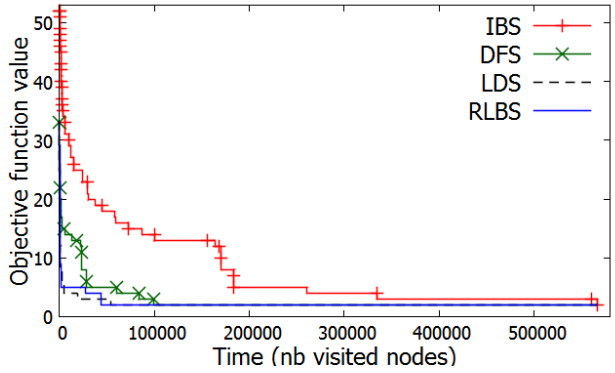


FIGURE 6 – La valeur de la fonction objectif en fonction du temps de calcul de RLBS, LDS, IBS et DFS sur un problème jouet

pas suffisamment de temps pour attendre la solution optimale, on voulait aussi considérer le temps nécessaire pour obtenir les solutions de qualités intermédiaires. Le tableau 2 montre, pour chaque cas, le temps de calcul moyen nécessaire pour obtenir une solution d'une qualité quelconque. La dernière colonne montre qu'en moyenne, pour tous les problèmes et toutes les qualités de solutions, on peut s'attendre à une amélioration de temps de calcul de 33.46%.

Enfin, on a généré des problèmes jouets (Fig. 6) afin de comparer RLBS à un autre algorithme pour lequel on n'a pas été capable de résoudre le problème original dans une durée de temps raisonnable (plus de 150 heures de traitement avec Choco v2.1.5). RLBS est comparé à d'autres approches qui ne modifient pas les heuristiques de branchement (DFS, LDS). On a choisi d'inclure les résultats de Impact-Based Search (IBS) [17] (qui ne nous permet pas d'employer nos heuristiques de choix de variable/valeur spécifiques) afin d'illustrer le fait que tenter d'apprendre à brancher au lieu d'utiliser de bonnes heuristiques de branchement déjà connues ne semble pas être une bonne idée (du moins pas dans le cas du problème étudié et IBS). IBS montre le pire résultat, probablement car il ne peut pas tirer profit des stratégies de branchement spécifiques connues d'être très efficaces pour ce problème. DFS est aussi surpassé par LDS, comme rapporté dans la littérature pour ce problème précédemment.

5 Conclusion

Nous avons proposé un simple mécanisme d'apprentissage basé sur l'apprentissage par renforcement qui permet au solveur d'apprendre dynamiquement à effectuer des retours arrière de manière efficace. Ceci a été évalué pour un problème industriel difficile de planification et ordonnancement, qui est uniquement résolu en utilisant des heuristiques de branchement spécifiques. La stratégie de retour arrière proposée améliore grandement la performance de l'exploration en comparaison avec LDS. Ceci grâce au mécanisme d'apprentissage qui permet d'identifier les nœuds vers lesquels il est le plus profitable d'effectuer des retours arrière.

L'utilisation de données réelles issues de l'industrie a montré la grande valeur de cette approche. Cependant, des questions demeurent concernant la performance de l'algorithme sur des problèmes pour lesquels on ne connaît pas de bonnes heuristiques de branchement. Dans cette situation, est-ce que ceci vaut la peine d'essayer d'identifier un meilleur nœud candidat au retour arrière ?

D'autres chercheurs [21] ont déjà utilisé l'apprentissage par renforcement pour apprendre à brancher. À notre connaissance, ceci est la première utilisation de l'apprentissage par renforcement pour apprendre une stratégie de retour arrière. Notre approche offre l'avantage qu'elle est utilisable peu importe la stratégie de branchement employée. Ceci la rend très efficace pour les problèmes pour lesquels une bonne straté-

gie de branchement est connue. Cependant, rien n'empêche de combiner les deux (apprendre une stratégie de branchement et une stratégie de retour arrière), ce qui pourra être fait dans le cadre de travaux futurs.

Remerciements

Ce travail a été soutenu par les partenaires industriels du Consortium de Recherche FORAC.

Références

[1] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2004.

[2] Andrew G Barto. *Reinforcement learning : An introduction*. MIT press, 1998.

[3] J Christopher Beck and Laurent Perron. Discrepancy-bounded depth first search. In *Proceedings of the Second International Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems (CPAIOR), Germany, Paderborn*, pages 7–17, 2000.

[4] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *ECAI*, volume 16, page 146, 2004.

[5] Justin A Boyan and Andrew W Moore. Using prediction to improve combinatorial optimization search. In *Sixth International Workshop on Artificial Intelligence and Statistics*, 1997.

[6] Jonathan Gaudreault, Pascal Forget, Jean-Marc Frayret, Alain Rousseau, Sebastien Lemieux, and Sophie D'Amours. Distributed operations planning in the softwood lumber supply chain : models and coordination. *International Journal of Industrial Engineering : Theory Applications and Practice*, 17 :168–189, 2010.

[7] Jonathan Gaudreault, Gilles Pesant, Jean-Marc Frayret, and Sophie D'Amours. Supply chain coordination using an adaptive distributed search strategy. *Journal of Systems, Man, and Cybernetics, Part C : Applications and Reviews, IEEE Transactions on*, 42(6) :1424–1438, 2012.

[8] Diarmuid Grimes and Richard J Wallace. Learning from failure in constraint satisfaction search. In *Learning for Search : Papers from the 2006 AAAI Workshop*, pages 24–31, 2006.

[9] William D Harvey and Matthew L Ginsberg. Limited discrepancy search. In *Proceedings of International Joint Conference on Artificial Intelligence (1)*, pages 607–615, 1995.

[10] Wafa Karoui, Marie-José Huguet, Pierre Lopez, and Wady Naanaa. YIELDS : A yet improved limited discrepancy search for CSPs. In *Proceedings of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 99–111. Springer, 2007.

[11] Manuel Loth, Michele Sebag, Youssef Hamadi, and Marc Schoenauer. Bandit-based search for constraint programming. In *Principles and Practice of Constraint Programming*, pages 464–480. Springer, 2013.

[12] Manuel Loth, Michele Sebag, Youssef Hamadi, Marc Schoenauer, and Christian Schulte. Hybridizing constraint programming and monte-carlo tree search : Application to the job shop problem. In *Learning and Intelligent Optimization*, pages 315–320. Springer, 2013.

[13] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2) :431–441, 1963.

[14] Victor V Miagkikh and William F Punch III. Global search in combinatorial optimization using reinforcement learning algorithms. In *Proceedings of Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1. IEEE, 1999.

[15] Thierry Moisan, Jonathan Gaudreault, and Claude-Guy Quimper. Parallel discrepancy-based search. In *Principles and Practice of Constraint Programming*, pages 30–46. Springer, 2013.

[16] Robert Moll, Andrew G Barto, Theodore J Perkins, and Richard S Sutton. Learning instance-independent value functions to enhance local search. In *Advances in Neural Information Processing Systems*. Citeseer, 1998.

[17] Philippe Refalo. Impact-based search strategies for constraint programming. In *Proceedings of Principles and Practice of Constraint Programming–CP 2004*, pages 557–571. Springer, 2004.

[18] Wheeler Ruml. *Adaptive tree search*. PhD thesis, Citeseer, 2002.

[19] Wheeler Ruml. Heuristic search in bounded-depth trees : Best-leaf-first search. In *Working Notes of the AAAI-02 Workshop on Probabilistic Approaches in Search*, 2002.

[20] Pascal Van Hentenryck. *The OPL optimization programming language*. Mit Press, 1999.

[21] Yuehua Xu, David Stern, and Horst Samulowitz. Learning adaptation to solve constraint satisfaction problems. In *Proceedings of Learning and Intelligent Optimization (LION)*, 2009.

[22] Wei Zhang and Thomas G Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of International Joint Conferences on Artificial Intelligence*, volume 95, pages 1114–1120. Citeseer, 1995.

Implémentation du problème de calepinage de façades avec Choco-Solveur

A. F. Barco [(1)] **J.G. Fages**[(2)] **É. Vareilles**[(1)] **M. Aldanondo**[(1)] **P. Gaborit**[(1)]

[(1)] Université de Toulouse, Mines d'Albi, Albi, France
[(2)] COSLING S.A., Nantes, France
[(1)]{abarocsa,vareille,aldanond,gaborit}@mines-albi.fr [(2)]jg.fages@gmail.com

Résumé

Nous présentons dans cette communication la suite des travaux présentés aux JFPC2014, concernant la génération de solutions de calepinage (ou pavage) de façades en vue de leur rénovation thermique. Une nouvelle version de cet algorithme de résolution à base de contraintes est proposée et implémentée avec Choco-Solver version 3.0. Le problème de calepinage de façades présente trois caractéristiques majeures. Premièrement, le nombre de panneaux rectangulaires et paramétrables à positionner en façade est inconnu avant résolution. Deuxièmement, les panneaux ne peuvent pas se superposer et doivent absolument contenir les nouvelles menuiseries (fenêtres, portes). Troisièmement, ils ne peuvent être accrochés en façade qu'à des endroits suffisamment résistants (traction et cisaillement) pour supporter la masse des panneaux (environ $500Kg.m^{-2}$). Seul l'aspect géométrique étant à considérer pour résoudre ce problème, nous le traitons donc comme un problème de *packing* en 2D. Pour ce faire, nous proposons une variante de la contrainte *Non-Overlap* adaptée à un nombre inconnu de panneaux paramétrables, ainsi qu'une heuristique de recherche dédiée pour limiter les incohérences et les *backtracks* liés à la contrainte de recouvrement des menuiseries par les panneaux. Cette variante de *Non-Overlap* est intégrée dans un prototype d'aide à la décision dédiés aux architectes maîtres d'oeuvre de la rénovation.

Abstract

We introduce a way of solving the problem of facade-layout synthesis using an open constraint programming environment. This problem inherits three characteristics from the industrial scenario : Its deals with the allocation of an unfixed number of rectangular panels that must not overlap, frames (existing windows and doors) must be overlapped by one and only one panel, and facades have specific areas providing certain load-bearing capabilities that allow to attach panels. Given the rectangular shape of panels and facades, the problem is treated as a two-dimensional packing problem. We show a variant of non-overlapping constraint for dealing with unfixed number of rectangles and a search heuristic that avoids the filtering of inconsistent values for frames mandatory overlapping. A prototype implemented in Choco-Solver is intended to assist architects decision-making in the context of building thermal retrofit.

1 Introduction

Actuellement, la consommation énergétique des bâtiments résidentiels et commerciaux représente plus du tiers de la consommation d'énergie dans les pays développés [5, 7, 17]. La réduction significative de cette part d'énergie consommée réside principalement dans la rénovation du parc immobilier existant, soit par une isolation intérieure, soit par une isolation extérieure [10]. Plusieurs techniques d'isolation peuvent être utilisées, comme celle consistant à recouvrir entièrement le bâtiment de panneaux multifonctionnels, rectangulaires et préfabriqués en usine [1, 8, 23]. Cependant, plusieurs difficultés doivent être soulevées lorsque l'on considère la rénovation par l'extérieur industrialisée : en général, une seule solution résulte du processus de conception, la conception de la solution est réalisée manuellement par les architectes et les maîtres d'oeuvre et le temps nécessaire à sa conception est conséquent (plusieurs jours de travail). Par conséquent, il est primordial d'assister la rénovation de bâtiments par l'extérieur par des outils informatiques d'aide à la décision [11].

Le calepinage des façades est le point crucial d'une rénovation énergétique performante. Celui-ci consiste en un pavage des façades (rectangulaires) avec un nombre indéterminé de panneaux paramétrables et rectangulaires. Un processus de calepinage consiste en

la détermination du nombre de panneaux à poser en façade, à leur dimensionnent (largeur, hauteur) et leur positionnement sur la façade en tenant compte des contraintes architecturales, géométriques et structurelles. Ce problème de calepinage s'apparente à un problème de *layout synthesis* [14] qui est par nature, un problème combinatoire [6, 24].

Trois aspects rendent ce problème de calepinage de façade stimulant et novateur [13, 14] :

- le nombre de panneaux à poser en façade et constituant la nouvelle enveloppe n'est pas *a priori* connu au début du processus de rénovation.
- Certains éléments constituant les façades doivent être entièrement couverts par les panneaux, ce qui est peu commun dans ce champ de recherche [14]. En effet, les menuiseries sont entièrement incluses dans les panneaux car montées directement en usine à l'intérieur de la structure de ces derniers.
- Les panneaux ne peuvent être fixés en façade qu'à des emplacements très spécifiques assez résistant pour supporter leur poids (nez-de-dalle, murs de refend, etc).

Au regard de la géométrie rectangulaires des panneaux et des façades, nous pouvons considérer ce problème comme un problème de *packing* à deux dimensions [9]. Nous pouvons donc appuyer nos propositions sur les travaux de ce domaine. De plus, au regard des contraintes manipulées, l'utilisation des approches par contraintes pour résoudre ce problème de calepinage de façades semble tout à fait indiquée [3, 4]. Les formes géométriques à manipuler dans notre problème étant uniquement rectangulaires, la contrainte géométrique `GEOST` [3] semble trop complexe et puissante à utiliser pour le résoudre. La contrainte globale de non recouvrement `DiffN` [4] semble mieux adaptée à nos besoins. Cependant, afin de guider efficacement la recherche de solutions, nous exploitons les possibilités de définition et d'implémentation de contraintes et de procédures de recherche ad'hoc. Par conséquent, nous considérons le solveur de contraintes comme le support au développement d'un outil d'aide à la décision de calepinage de façades. Néanmoins, ne pas connaître en avance le nombre de panneaux à poser en façades est un véritable problème car une grand majorité des environnements de programmation par contraintes implémente des contraintes globales et des moteurs de résolution portant sur un ensemble connu et défini de variables.

Raisonner sur un problème de contraintes à structure variable et dynamique, i. e., ajout de contraintes et de variables en cours de résolution, est un domaine toujours en cours d'étude en programmation par contraintes. Dans [2], les auteurs résolvent le problème du nombre inconnu de variables par l'ajout de variables en cours de résolution. En soi, cela revient à désactiver et activer des contraintes en fonction des variables qu'elles relient. Néanmoins, si l'idée semble simple et séduisante, son implémentation est très délicate. Nos travaux s'inspirent donc plutôt de [22], où les auteurs introduisent la notion de contrainte globale ouverte ou *open global constraint*.

Une contrainte globale ouverte est une extension d'une contrainte globale existante qui inclut un ensemble de variables (ou un vecteur de variables binaires) pour indiquer le sous-ensemble de variables à considérer dans le traitement de la contrainte. En d'autres termes, certaines variables de décision du problème sont optionnelles (voir [12, 20] et Section 4.4.16 en [19] pour plus d'informations).

L'objectif de notre article est de proposer une solution au problème de calepinage de façades considéré comme un problème de *packing* à deux dimensions avec rectangles optionnels. Pour cela, nous utilisons la variante ouverte de la contrainte `DiffN` [4] pour gérer les rectangles ou panneaux qui appartiennent potentiellement à une solution. Nous présentons dans cet article une heuristique de recherche simple mais qui capture bien la structure de notre problème. Les solutions proposées ont implémentées dans l'environnement *open-source* Choco [18]. Le document est divisé comme suit. Dans la section 2, les éléments du problème de calepinage de façades sont introduits. Dans la section 3, la formalisation de ce problème comme un problème de satisfaction de contraintes est présentée. Dans la section 4, nous fournissons des détails techniques de notre implémentation. Dans la section 5, une heuristique de recherche qui capture la structure du problème est présentée. Ensuite, dans la section 6, nous montrons une certaine évaluation de la performance de nos méthodes. Enfin, certaines conclusions sont discutées dans la section 7.

2 Éléments de la rénovation

Façades. Une façade est représentée par un plan à deux dimensions (voir figure 1), dont le repère (0,0) se situe dans le coin en bas à gauche. Une façade contient des zones rectangulaires définissant :

- le périmètre de la façade avec ses dimensions (hauteur et largeur).
- les menuiseries existantes (portes et fenêtres) qui jouent un rôle important dans la recherche de solutions car elles doivent être totalement recouvertes par un et un seul panneau. Les menuiseries sont définies par :
  - la position (x,y) du coin en bas à gauche relative à la façade,

— une hauteur et une largeur en mètres.
— des zones de support où les panneaux peuvent être accrochés. Ces zones de support doivent être assez solides pour supporter la masse des panneaux rapportés en façade. Les zones de support sont définies par :
  — la position (x,y) du coin en bas à gauche relative à la façade,
  — une hauteur et une largeur en mètres,

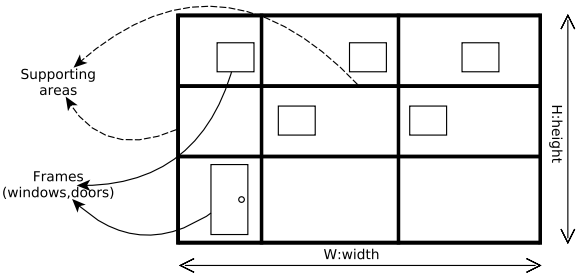


FIGURE 1 – Façades : menuiseries et zones de support.

Panneaux rectangulaires. Les panneaux rectangulaires, comme illustré en figure 2, sont paramétrables et peuvent contenir plusieurs équipements (menuiseries, volets, etc). Ces panneaux sont paramétrés les uns après les autres dans le processus de calepinage et sont fabriqués en usine et montés sur site dans un ordre bien défini. Les panneaux rectangulaires paramétrables sont définis par :

— des dimensions (hauteur, largeur) qui sont fonction du fabricant de panneaux,
— un type d'isolant et une épaisseur d'isolant qui caractérisent la performance énergétique du panneau, et qui dépendent du fournisseur de panneau,
— la liste des nouvelles menuiseries qu'ils contiennent. Chaque menuiserie est caractérisée par une position (x,y) relative au panneau et une encombrement dans le panneau. Une distance minimale Δ doit être respectée entre les bords des menuiseries et du panneau qui les inclut,
— un coût dépendant principalement de la dimension du panneau et des équipements qu'il contient (en €),
— une performance énergétique (en watt par mètre carré par kelvin ($w.m^{-2}.k^{-1}$) dépendant principalement, de ses dimensions, du type d'isolant et de son épaisseur.

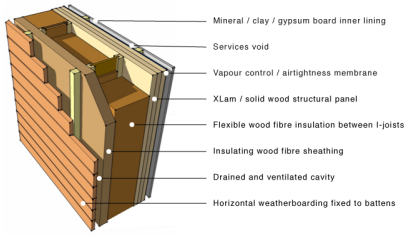


FIGURE 2 – panneaux rectangulaires paramétrables.

Éléments du problème. Comme indiqué en introduction, trois aspects rendent ce problème original et intéressant, le nombre de panneaux à poser en façade étant le plus problématique. La figure 3, partie (1) présente trois panneaux qui ne respectent pas les contraintes évoquées : le panneau $P1$ est en collision avec une menuiserie, le panneau $P2$ ne respecte pas la distance minimale Δ entre le bord de la menuiserie et son bord gauche, et le panneau $P3$ ne peut pas être accroché en façade car il n'est pas aligné avec une zone de support. La partie (2) de la figure 3, présente une solution de calepinage composée de 6 panneaux consistants avec les contraintes évoquées.

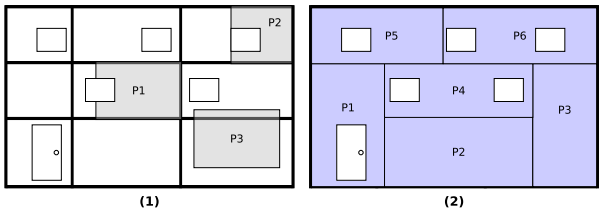


FIGURE 3 – Exemple de calepinage avec panneaux adéquats et non adéquats.

Restrictions. Nous posons les restrictions suivantes sur le problème présenté dans cet article. Tout d'abord, les zones de support sont suffisamment résistantes pour supporter le poids des panneaux quelque soient leurs dimensions, le type d'isolant et l'épaisseur d'isolant. Deuxièmement, pour estimer le coût et la performance énergétique atteinte après rénovation, nous considérons que l'épaisseur des panneaux est constante et qu'il n'y a qu'un seul type d'isolant. Et troisièmement, nous considérons que le coût et la performance énergétique dépendent uniquement et sont proportionnels à la surface en m^2 des panneaux.

2.1 Fonction de coût

Les différentes solutions de calepinage sont ordonnées en fonction de deux critères qui sont le coût et la performance énergétique. Le coût d'une solution dépend principalement du coût des panneaux posés et de leurs équipements. Dans le cadre de cet article, nous ne tenons pas compte du coût des équipements car celui-ci n'affecte pas le calepinage final. Le coût d'une solution dépend donc uniquement de celui des panneaux et en particulier de sa surface en m^2, en tenant compte des coûts des matières premières (identique pour chaque solution en vu des restrictions posées) et du coût de fabrication. Les experts nous ont fourni un moyen de calculer le coût d'une enveloppe d'isolation par rapport à la taille de panneaux. La formule 1 exprime cette connaissance

$$c_{fac} = \sum_{i=1}^{N}(w_i \times h_i) + (\alpha - w_i - h_i) \qquad (1)$$

où w_i et h_i représentent respectivement la largeur et la hauteur du panneau i, et α est un facteur fourni par le fabricant qui dépend du procédé de fabrication en question. Nous pouvons observer dans la formule 1 que le terme $(\alpha - w_i - h_i)$ diminue avec la taille du panneau. La fabrication de grands panneaux devient alors moins cher que celle de petits panneaux (en coût de main d'oeuvre principalement).

Maintenant, d'un point de vue de performance thermique, des grands panneaux minimisent les joints de l'enveloppe. En fait, en raison des caractéristiques thermiques de la rénovation, il est mieux de réduire le nombre de joints sur une enveloppe : C'est au niveau des jonctions entre panneaux que se font les transferts thermiques. Par conséquent, la performance d'un plan de masse dépend de la longueur de la jonction entre deux panneaux consécutifs. Le calcul de la longueur des jonctions pour une enveloppe donnée est simple, la formule 2 exprime cette connaissance :

$$ttc_{fac} = w_{fac} + h_{fac} + \sum_{i=1}^{N}(w_i + h_i) \qquad (2)$$

où ttc_{fac} est le *coefficient de transfert thermique* de la façade fac, w_{fac} et h_{fac} sont respectivement la largeur et la hauteur de la façade, w_i et h_i représentent respectivement la largeur et la hauteur du panneau i et N est le nombre de panneaux qui composent l'enveloppe. Dans le cadre d'une rénovation par l'extérieur, celles-ci apparaissent principalement à la jonction de plusieurs panneaux. Par conséquent, les façades doivent porter le moins de panneaux possibles afin de limiter ces fuites thermiques, tout en respectant les contraintes architecturales, des zones d'accroche et des contraintes de fabrication.

3 Calepinage de façades et CSP

Dans cette section, nous formalisons le problème de calepinage de façades comme un problème de satisfaction de contraintes. Un problème de satisfaction de contraintes ou CSP se définit à l'aide d'un ensemble de variables $\mathcal{V}$, chacune associée à un domaine de valeurs possibles $\mathcal{D}$ et un ensemble de relations ou contraintes $\mathcal{C}$ portant sur ces variables [15, 16]. La solution d'un CSP se définit par l'assignation à chaque variable d'une valeur unique de son domaine de valeurs possibles, tel que l'ensemble des contraintes de $\mathcal{C}$ soit satisfait.

Posons maintenant la notation suivante : F correspond à l'ensemble des menuiseries, et S à l'ensemble des zones de support. Soit $o_{e.d}$ et $l_{e.d}$, respectivement l'origine et la longueur d'un élément e dans une direction d, avec $d \in \{1, 2\}$. Par exemple, $o_{fr.1}$ correspond à l'origine selon l'axe horizontal et $l_{fr.1}$ correspond à la largeur d'une menuiserie fr (axe horizontal). De plus, lb_d et ub_d correspondent respectivement à la dimension minimale et maximale, selon une direction d d'un panneau.

3.1 Variables

Au regard de notre problème de calepinage, nous pouvons facilement identifier que les variables de décision portent sur l'appartenance ou non des panneaux à une solution. La principale difficulté réside dans la formalisation de ce problème comme un CSP avec un nombre inconnu de panneaux constituant une solution. Par conséquent, nous introduisons une borne maximale sur le nombre de panneaux potentiellement à utiliser pour construire une solution. Soit n ce nombre maximum de panneaux estimés pour couvrir une façade. En considérant les dimensions minimales et maximales des panneaux, nous pouvons borner n par la valeur suivante : $n = \left\lceil \frac{l_{fac.1} \times l_{fac.2}}{lb_1 \times lb_2} \right\rceil$.

Nous créons alors un ensemble de n variables de panneaux optionnels, chacune se référant à un panneau et indiquant si celui-ci appartient ou non à la solution. Chaque panneau $0 \leq p \leq n$ est décrit par son appartenance à la solution, son origine et ses dimensions :

- $b_p \in \{0, 1\}$ indique si le panneau appartient ou non à la solution,
- $o_{p.d} \in [0, l_{fac.d}]$ correspond à l'origine du panneau p dans la direction d.
- $l_{p.d} \in [lb_{p.d}, ub_{p.d}]$ correspond à la longueur du panneau p dans la direction d.

Nous pouvons noter que comme un rectangle ou panneau est déjà défini par un tableau de variables entières (ses coordonnées et ses dimensions), il est plus naturel d'étendre celui-ci avec une cinquième variable binaire représentant son appartenance à la solution, que d'introduire des variables pour représenter tous les rectangles à considérer [22].

Nous introduisons la variable objectif z représentant le nombre de rectangles qui appartiennent à la solution. Nous avons donc $\left\lceil \frac{fac_1 \times fac_2}{ub_1 \times ub_2} \right\rceil \leq z \leq n$

3.2 Contraintes métier

Pour définir le calepinage d'une façade, nous posons les cinq contraintes métier suivantes entre les éléments du problème :

(C1) *Contraintes de fabrication et de livraison sur chantier portant sur les dimensions minimales* `lb` et maximales `ub` *autorisées des panneaux selon une ou deux directions.*

$$\forall p, 0 \leq p < n, d \in \{1,2\}, \quad lb_d \leq l_{p.d} \leq ub_d$$

(C2) *Le bas des panneaux doit être aligné sur une zone de support afin d'y être accroché.*

$$\forall p, 0 \leq p < n, \exists s \in S, d \in \{1,2\} \mid \\ o_{s.d} \leq o_{p.d} \vee o_{p.d} + l_{p.d} \leq o_{s.d} + l_{s.d}$$

(C3) *La façade doit être entièrement recouverte de panneaux.*

$$\sum_{p=0}^{n-1} \left(b_p \times l_{p.1} \times l_{p.2}\right) = l_{fac.1} \times l_{fac.d}$$

Nous devons souligner que ces contraintes engendrent un filtrage très faible, car le domaine des variables l peut être grand et la contrainte permet aux panneaux de se chevaucher. Par conséquent, celui-ci doit être renforcé par l'heuristique de recherche.

(C4) Deux panneaux appartenant à la même solution ne doivent pas se chevauche.

$$\texttt{OpenDiffN}(b, o, l)$$

(C5) *Chaque menuiserie d'une façade doit être recouverte par un et un seul panneau. De plus, une distance minimale* Δ *doit être respectée entre le bord d'une menuiserie et le bord le plus près du panneau qui l'inclut.*

$$\forall_f \in F, \exists p, 0 \leq p < n, d \in \{1,2\} \mid \\ o_{p.d} + \Delta \leq o_{f.d} \wedge o_{f.d} + l_{f.d} \leq o_{p.d} + l_{p.d} + \Delta$$

3.3 Contraintes de suppression des symétries

(C6) Afin d'éviter les symétries, nous appliquons une contrainte d'ordre lexicographique sur la présence et l'origine des panneaux [21] .

$$\texttt{LexChainLessEq}(\{\{(1-b_p), o_{p,1}, o_{p,2}\} \mid \\ 0 \leq p < n\})$$

Cette contrainte permet d'assurer qu'une priorité est donnée aux premiers panneaux et que les panneaux utilisés dans une solution sont ordonnés de manière géométrique.

(C7) Afin d'éviter les pertes de temps sur le paramétrage (position et dimensions) des panneaux non utilisés dans une solution, nous valuons les variables de leur origine au premier point d'accroche et leurs dimensions à leurs valeurs minimales. En supposant que le premier point d'accroche valide correspond à l'origine de la façade (0,0), nous avons la contrainte suivante :

$$\forall p, 0 \leq p < n, \forall d \in \{1,2\}, \\ b_p = 0 \Rightarrow o_{p,d} = 0 \wedge l_{p,d} = lb_d$$

4 Implémentation

Cette section donne des détails sur l'implémentation du modèle. Fondamentalement, notre solution suit l'approche de [22], où une variable ensembliste est utilisée pour gérer les variables de décision qui sont potentiellement dans la solution. Cependant, dans ce travail, comme les rectangles sont déjà composés de plusieurs attributs entiers, nous avons trouvé plus naturel d'utiliser une variable binaire supplémentaire par rectangle à la place d'une variable ensembliste. Intuitivement, une contrainte ouverte avec des variables booléennes peut être implémentée avec des algorithmes de filtrage traditionnels et peut être améliorée en ciblant la structure du problème.

4.1 Une contrainte ouverte rectangle non-chevauchement (C4)

À notre connaissance, la contrainte `OpenDiffN` a déjà été implémentée (voir `non-chevauchement` des rectangles optionnels à la section 4.4.16 de [19] par exemple) mais nous considérons qu'il est nécessaire de donner une brève description de son comportement. L'algorithme de filtrage de la contrainte `OpenDiffN` vérifie si deux panneaux qui font partie de la solution, c'est à dire dont le b_i est égal à 1, ne se chevauchent pas, et procède à un filtrage de domaine pour éviter les chevauchements, comme des propagateurs de `DiffN` traditionnels font. Réciproquement, si deux panneaux se chevauchent dans l'espace, alors le filtrage de domaine sur les variables booléennes fait en sorte qu'au moins un des ces panneaux n'est pas utilisé. Le filtrage global est renforcé par un algorithme de disjonction constructive qui calcule un support valide (du point

de vue du calepinage) pour le coin en bas à gauche de chaque rectangle.

4.2 Une contrainte dédiée à couvrant menuiseries (C5)

La contrainte de C5 est propagée en utilisant une approche dédiée. L'algorithme de filtrage est assez simple et fonctionne comme suit : pour chaque menuiserie, deux panneaux de *support* (tels que la menuiserie est inscrit dans les panneaux) sont calculés. Si aucun panneau de support n'est trouvé, alors le solveur échoue. Dans le cas où un seul panneau est trouvé, alors une procédure de filtrage est appliquée au panneaux pour couvrir la menuiserie. Enfin, dans le cas où deux panneaux sont trouvés, alors il n'y a pas de propagation parce que nous ne savons pas quel panneau sera utilisé pour couvrir la façade.

4.3 Une contrainte liée à la suppression des symétries

La contrainte lexicographique permet de générer un ensemble de solutions minimal : les panneaux étant tous identiques et positionnables n'importe où sur la façade, des solutions identiques en terme de géométrie mais composées de panneaux différents peuvent être générées. Afin d'éviter cela, nous utilisons une contrainte lexicographique permettant d'assurer que les panneaux sont utilisés dans un ordre prédéfini. Un panneau P_i ne peut pas être positionné en façade, ni appartenir à la solution, si l'ensemble des panneaux le précédant dans l'ordre lexicographique ne sont pas tous utilisés et positionnés : $\forall_{k=1}^{n}\ (k < i) \wedge (P_k.b_p = 1)$.

Nous allons maintenant voir comment exploiter la structure du problème au sein de la recherche.

Algorithm 1: Heuristique de recherche dédié.

```
1  def Fonction obtenirDecisionBranchement:
2      int r ← −1 ; // calculer le premier rectangle non
       fixé;
3      for i ← 0 to n do
4          if |dom(b_i)| + |dom(o_i1)| + |dom(o_i2)| +
           |dom(o_l1)| + |dom(o_l2)| > 5 then
5              r ← i ; break;
6      if r == −1 then
7          return null ; // tous les rectangles sont fixés
           (une solution a été trouvée)
8      // Trouver l'affectation de variable-valeur
       suivante pour effectuer
9      VarEntière var, Entière val;
10     if |dom(b_i)| > 1 then
11         var ← b_i;
12         val ← 1;
13     else if |dom(o_i1)| > 1 then
14         var← o_i1;
15         val ← dom(o_i1).lb;
16     else if |dom(o_i2)| > 1 then
17         var← o_i2;
18         val ← dom(o_i2).lb;
19     else if |dom(o_i1)| > 1 then
20         var← l_i1;
21         val ← dom(l_i1).ub;
22     else
23         var← l_i2;
24         val ← dom(l_i2).ub;
25     Decision d = new Decision(var, val);
26     if var == o_i1 ∨ var == o_i2 then
27         d.once() ; // empêche le solveur d'essayer
           différentes valeurs sur marche arrière
28     return d;
```

5 L'heuristique de recherche

L'heuristique de recherche est responsable de la délimitation des variables de décision lorsque la propagation ne peut déduire plus d'informations. Notre heuristique est décrite dans l'algorithme 1. C'est une approche constructive qui considère chaque rectangle un par un et utilise la priorité de sélection variable suivante : b_i, o_{i1}, o_{i2}, l_{i1}, et enfin l_{i2}. L'originalité de cette méthode est que certaines décisions ne peuvent pas être niées : instruction `d.once()` dans la ligne 27 indique au solveur de ne pas essayer différentes valeurs en cas d'échec. Par exemple, si $o_1 = 1$ et le nœud échoue, il ne sera pas essayer de propager $o_1 \neq 1$ et de calculer une nouvelle décision. Au lieu de cela, il va revenir en arrière une fois de plus (de la décision associé à la taille du rectangle précédent). Notre algorithme est complet et explore l'ensemble de l'espace de recherche afin de déterminer l'ensemble des solutions.

L'heuristique met en œuvre les choix de conception clés suivants :

1. Nous choisissons d'assigner la valeur 1 à b_i afin d'arriver plus rapidement à une solution.
2. Les variables de position o_1 et o_2 sont fixées à leurs limites inférieures afin de ne pas laisser les lieux découverts entre le rectangle examiné et les rectangles placés précédemment. En bref, les variables réelles de décision sont b, l_1 et l_2. Mais o_1 et l_2 devraient être fixées de manière déterministe, sans *backtracking*. Comme les rectangles sont ordonnés, essayer une valeur supérieure conduirait à un trou sur la façade, ce qui est interdit. Notez que ce n'est possible uniquement que parce

que le filtrage est assez fort : la borne inférieure est en effet un support valide du point de vue du calepinage.

3. Les variables de taille l_1 et l_2 sont fixées à leurs limites supérieures pour essayer des rectangles aussi grands que possible et ainsi couvrir la plus grande surface possible (en m^2. Ceci permet d'obtenir une première solution qui est très proche de la valeur optimale.

Dans l'ensemble, le mécanisme de recherche combine deux principes différents. Premièrement, il est basé sur une approche constructive glouton qui est efficace mais limitée. Deuxièmement, il utilise un algorithme de *backtracking* personnalisé (certaines décisions ne peuvent pas être niées) pour explorer des alternatives.

6 Évaluation

Dans cette section, nous évaluons empiriquement notre modèle, qui a été implémenté en Java, avec le solveur `Choco` [18]. Nous considérons un ensemble de 20 cas, généré à partir des spécifications réalistes. La surface totale de la façade varie de 10^4 à 10^6 pixels (le rapport en la taille des pixels et la surface réelle n'est pas considérée) pour tester l'extensibilité de l'approche. Des panneaux sont générés en utilisant une borne inférieure de 20 (pixels) et une limite supérieure de 150 (pixels), à la fois en largeur et en hauteur.

6.1 Une approche en deux étapes

Dans une première expérience on veut évaluer si le nombre maximal de panneaux utilisés est une bonne approximation de l'optimum. La figure 4 présente le nombre de panneaux utilisés et le nombre de panneaux optionnels pour chaque instance. Le nombre maximum de panneaux, qui représente le pire des cas, où la taille des bornes inférieures de panneaux sont utilisées, n'est jamais atteint. Nous pouvons voir que cette valeur est en fait très loin de l'optimum. En outre, ce nombre maximum est une limite supérieure trop haute : pour une façade de la taille 2300×575, le solveur gère 3220 panneaux optionnels pour calculer une première solution qui utilise seulement 96 panneaux. Cela signifie que nous créons trop de variables inutiles qui ralentissent le processus de résolution. Par conséquent, nous mettons en place une approche en 2 étapes : d'abord une solution est calculée en utilisant notre modèle. Ensuite, nous créons un nouveau modèle avec la valeur de la solution précédente comme le nombre maximum de panneaux. Puis, nous énumérons toutes les solutions optimales.

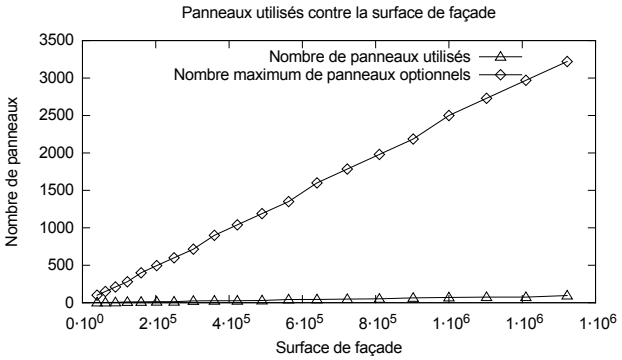


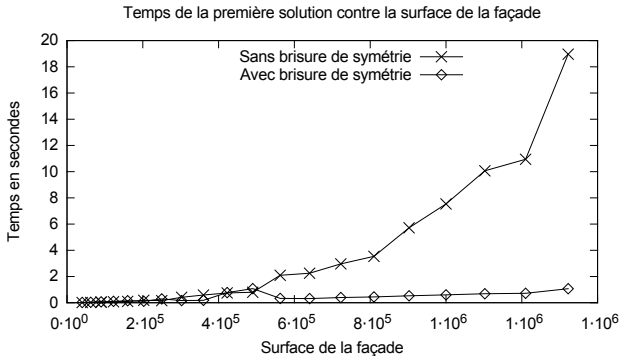
FIGURE 4 – Le nombre maximum de panneaux en option et le nombre de panneaux utilisés dans la première solution, pour chaque instance.

6.2 Impact de suppression des symétries

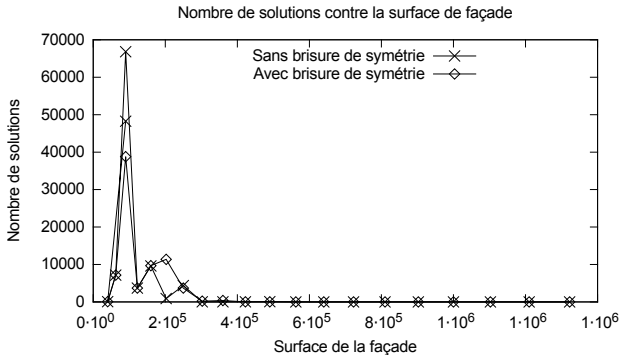
Dans une seconde expérience, nous mesurons l'impact de l'ajout de contraintes de suppression de symétries. Plus précisément nous comparons le temps de trouver une première solution (Figure 5.a) et le nombre de solutions calculées (Figure 5.b) avec et sans contraintes de suppression de symétries. En raison de l'énorme quantité de solutions, nous utilisons un délai de 60 secondes. Comme nous pouvons le voir sur la figure 5, les contraintes de suppression de symétries accélèrent la recherche. En outre, elles permettent de sauter des milliers de solutions qui sont identiques pour l'utilisateur final. C'est encore plus important que de gagner du temps de calcul. Notez qu'il semble que le nombre de solutions diminue lorsque la surface de façade augmente : c'est à cause de la limite de temps. Comme le problème devient plus gros, le processus de résolution devient plus lent et énumère moins de solutions en un temps donné.

6.3 Comparaison de recherche

En ce qui concerne différentes heuristiques de recherche, aucune n'est adapté pour résoudre le problème. Des stratégies de recherche boîte noire, bien connues tels que domOverWDeg, recherche a base d'impact ou la recherche par activité, ne fonctionnent pas bien compte tenu des particularités du problème. Ces heuristiques sont conçues pour résoudre les problèmes d'une manière aveugle, quand nous n'avons pas connaissance d'experts du problème. Dans notre cas, nous mélangeons des variables de genre très différent (les booléens, les positions, tailles) que nous sommes en mesure d'ordonner et de grouper par des rectangles. L'introduction aléatoire soit sur la sélection de variables soit sur la sélection de valeur peut être désastreuse. En particulier, en utilisant des valeurs ar-



(a)



(b)

FIGURE 5 – Temps nécessaire pour atteindre une première solution et le nombre de solutions calculées, avec un délai de 60 secondes.

bitraires pour o_1 et o_2 fait une énorme quantité de possibilités pour les places non couvertes.

Néanmoins, afin de valider la pertinence de notre heuristique dédiée, nous avons testé 16 heuristiques prédéfinies de `Choco` sur une petite façade (400×100). Nous présentons les résultats pour celles qui ont obtenu au moins une solution. Ces stratégies sont : `domOverWDeg` qui sélectionne la variable en fonction du réseau de contraintes ; `lexico_LB` qui choisit la première variable non instanciée, et l'assigne à sa borne inférieure ; `lexico_Split` qui choisit la première variable non instanciée, et supprime la deuxième moitié de son domaine ; `maxReg_LB` qui choisit la variable non instanciée avec la plus grande différence entre les deux plus petites valeurs dans son domaine, et l'assigne à sa borne inférieure ; `minDom_LB` qui choisit la première variable non instanciée avec la plus petite taille de domaine, et l'assigne à sa borne inférieure et ; `minDom_MidValue` qui choisit la première variable non instanciée avec la plus petite taille de domaine, et l'assigne à la valeur plus proche du milieu de son domaine. La dernière entrée est notre propre heuristique. Rappelons que les variables sont ordonnées b, o_1, o_2, l_1 et l_2.

Les tableaux 1 et 2 donnent respectivement les résultats obtenus pour un exemple 400×100 et 400×200. Bien que certaines heuristiques prédéfinies ont une bonne performance sur le premier (petit) exemple, aucun d'eux n'est extensible. En fait, aucune heuristique de recherche prédéfini trouve une solution pour une façade avec la taille 400×200 en un temps de calcul raisonnable alors que notre heuristique dédiée trouve déjà 726 solutions différentes en 180 secondes. Notre heuristique surpasse clairement les autres. Plus important encore, elle permet de toujours proposer une solution rapidement, ce qui est essentiel pour l'utilisateur.

TABLE 1 – Comparaison heuristique sur une façade 400×100.

Stratégie	TPS (s)	Total (s)	#nœuds	#sols
domOverWDeg	18.77	19.94	1412897	66
lexico_LB	**0.03**	0.22	2380	66
lexico_Split	**0.03**	**0.16**	**441**	66
maxReg_LB	**0.03**	0.22	2380	66
minDom_LB	0.74	19.96	1411183	66
minDom_MidValue	43.43	47.32	4755206	66
dedicated	**0.03**	0.85	10978	66

TABLE 2 – Comparaison heuristique sur une façade de 400×200 avec un délai de 3 minutes.

Stratégie	TPS (s)	#nœuds	#sols
domOverWDeg	-	7286594	0
lexico_LB	-	5772501	0
lexico_Split	-	4966920	0
maxReg_LB	-	5490088	0
minDom_LB	-	11961712	0
minDom_MidValue	-	11157755	0
dedicated	**0.039**	**3499527**	**726**

7 Conclusions

La maîtrise de la consommation énergétique des bâtiments est l'un des défis majeurs du 21^e siècle. La réduction de la consommation énergétique des bâtiments se concentre maintenant sur la rénovation de bâtiments existants. Nos travaux s'inscrivent dans le cadre du projet ADEME C.R.I.B.A. qui vise à industrialiser la rénovation par l'extérieur d'immeubles de logements collectifs pour atteindre une performance énergétique proche de $25kWh/m^2/an$.

Cet article présente une solution au problème du calepinage de façades. Ce problème de calepinage est considéré comme un problème de packing à deux dimensions avec rectangles optionnels. Bien qu'il existe un grand corps de la littérature sur packing à deux dimensions, notre scénario industriel comporte trois caractéristiques jamais considérées simultanément : l'attribution d'un nombre quelconque de panneaux rectangulaires qui ne doivent pas se chevaucher, certains

éléments de la façade (les menuiseries) doivent absolument être contenues dans les panneaux, et les façades ont des zones spécifiques pour fixer les panneaux. Ainsi, autant que nous le savons, aucun système de soutien, ni système de conception est bien adapté pour traiter ces particularités.

Considérant que le nombre de panneaux n'est pas connu à l'avance, nous avons utilisé une variante de la contrainte de `DiffN` pour gérer rectangles optionnels par des variables booléennes. Nous avons mis en place une contrainte pour le chevauchement obligatoire des menuiseries et une heuristique de recherche dédié qui tire parti de la structure du problème et est en mesure d'énumérer des solutions optimales par rapport le nombre de panneaux utilisés. Nos solutions proposées ont été implémentés avec le solveur `Choco` et de démontrer la validité de notre méthode. En particulier, notre modèle prend l'avantage à la fois d'une approche glouton et une approche de recherche d'arbre pour sortir *plusieurs* bonnes solutions très rapidement, ce qui était la condition la plus critique du client. Cela souligne l'importance de la grande souplesse de programmation par contraintes.

Plusieurs pistes restent à envisager : la définition d'autres heuristiques de recherche incluant des critères esthétiques, la prise en compte des contraintes de masse des panneaux afin que ceux-ci puissent effectivement être fixés en façade et la prise en compte de l'ensemble de variables du problème (type et épaisseur d'isolant) qui ont un impact sur la définition des solutions. La fonction objectif doit être modifiée pour prendre en compte le prix de la solution et une estimation de la performance énergétique du bâtiment après rénovation. Un benchmark sur différentes géométries de bâtiments et avec plusieurs heuristiques doit aussi être conduit afin de confronter nos propositions aux travaux déjà existants.

La programmation par contraintes est tout à fait adaptée à la résolution de ce problème de calepinage car, d'une part, une fonction objectif doit être minimisée (nombre de panneaux définissant la solution de calepinage), et d'autre part, la conception d'un tel prototype d'aide à la décision à base de contraintes globales ouvertes est très facile grâce aux contraintes globales et aux algorithmes de recherche prédéfinis. La formalisation de ce problème comme un CSP a été réalisée par quatre personnes et a durée plusieurs mois (recueil, validation et formalisation des connaissances) quand l'implémentation des heuristiques de recherche dans un environnement de programmation par contraintes (i.e. `Choco`, `Gecode`, module de domaine fini de `Mozart-Oz`) a été réalisée par deux personnes. Le développement de l'outil a été réalisé en deux semaines de travail effectif.

Remerciements

Les auteurs souhaitent remercier leurs partenaires industriels *TBC Générateur d'Innovation*, *Millet* et *SYBois* ainsi que l'ensemble des membres du consortium de projet C.R.I.B.A. pour la construction du premier modèle à base de contraintes supportant cette problématique de calepinage. Ils tiennent aussi à remercier l'implication de *Cosling S.A.S.* pour le développement de la première maquette avec Choco solver.

Références

[1] A. F. Barco, E. Vareilles, M. Aldanondo, P. Gaborit, and M. Falcon. Calepinage à base de contraintes : application à la rénovation de bâtiments à haute performance énergétique. In *10th Journées Francophones de Programmation par Contraintes.*, pages 293–300. Association Française pour la Programmation par Contraintes, June 2014.

[2] Roman Barták. Dynamic global constraints in backtracking based environments. *Annals of Operations Research*, 118(1-4) :101–119, 2003.

[3] N. Beldiceanu, M. Carlsson, E. Poder, R. Sadek, and C. Truchet. A generic geometrical constraint kernel in space and time for handling polymorphic k-dimensional objects. In Christian Bessière, editor, *Principles and Practice of Constraint Programming - CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, pages 180–194. Springer Berlin Heidelberg, 2007.

[4] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Emmanuel Poder. New filtering for the cumulative constraint in the context of non-overlapping rectangles. *Annals of Operations Research*, 184(1) :27–50, 2011.

[5] The Energy Conservation Center. *Energy Conservation Handbook*. The Energy Conservation Center, Japan, 2011.

[6] P. Charman. Solving space planning problems using constraint technology, 1993.

[7] U.S. Green Building Council. *New Construction Reference Guide*. 2013.

[8] M. Falcon and F Fontanili. Process modelling of industrialized thermal renovation of apartment buildings. *eWork and eBusiness in Architecture, Engineering and Construction*, pages 363–368, 2010.

[9] S. Imahori, M. Yagiura, and H. Nagamochi. *Practical algorithms for two-dimensional packing.*

Chapter 36 of Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series), 2007.

[10] Bjørn Petter Jelle. Traditional, state-of-the-art and future thermal building insulation materials and solutions - properties, requirements and possibilities. *Energy and Buildings*, 43(10) :2549 – 2563, 2011.

[11] Yi-Kai Juan, Peng Gao, and Jie Wang. A hybrid decision support system for sustainable office building renovation and energy performance improvement. *Energy and Buildings*, 42(3) :290 – 297, 2010.

[12] Philippe Laborie. Ibm ilog cp optimizer for detailed scheduling illustrated on three problems. In Willem-Jan van Hoeve and JohnN. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 148–162. Springer Berlin Heidelberg, 2009.

[13] Kyoung Jun Lee, Woo Kim Hyun, Jae Kyu Lee, and Tae Hwan Kim. Case- and constraint-based project planning for apartment construction. *AI Magazine*, 19(1) :13–24, 1998.

[14] Robin S Liggett. Automated facilities layout : past, present and future. *Automation in Construction*, 9(2) :pp. 197 – 215, 2000.

[15] U. Montanari. Networks of constraints : Fundamental properties and applications to picture processing. *Information Sciences*, 7(0) :95 – 132, 1974.

[16] Carlos Olarte, Camilo Rueda, and FrankD. Valencia. Models and emerging trends of concurrent constraint programming. *Constraints*, 18(4) :535–578, 2013.

[17] Luis Pérez-Lombard, José Ortiz, and Christine Pout. A review on buildings energy consumption information. *Energy and Buildings*, 40(3) :394 – 398, 2008.

[18] C. Prud'homme and JG. Fages. An introduction to choco 3.0 an open source java constraint programming library. In *CP Solvers : Modeling, Applications, Integration, and Standardization. International workshop.*, Uppsala Sweden, 2013.

[19] Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. Modeling and programming with gecode, 2010.

[20] Andreas Schutt, Thibaut Feydy, and PeterJ. Stuckey. Scheduling optional tasks with explanation. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 628–644. Springer Berlin Heidelberg, 2013.

[21] Willem Jan van Hoeve and John N. Hooker, editors. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings*, volume 5547 of *Lecture Notes in Computer Science*. Springer, 2009.

[22] Willem-Jan van Hoeve and Jean-Charles Régin. Open constraints in a closed world. In J.Christopher Beck and BarbaraM. Smith, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3990 of *Lecture Notes in Computer Science*, pages 244–257. Springer Berlin Heidelberg, 2006.

[23] E. Vareilles, A.F. Barco Santa, M. Falcon, M. Aldanondo, and P. Gaborit. Configuration of high performance apartment buildings renovation : A constraint based approach. In *Industrial Engineering and Engineering Management (IEEM), 2013 IEEE International Conference on*, pages 684–688, Dec 2013.

[24] Machi Zawidzki, Kazuyoshi Tateyama, and Ikuko Nishikawa. The constraints satisfaction problem approach in the design of an architectural functional layout. *Engineering Optimization*, 43(9) :pp. 943–966, 2011.

Contraintes sur des flux appliquées à la vérification de programmes audio

Anicet Bart[1] **Charlotte Truchet**[2] **Éric Monfroy**[2]

LINA - Laboratoire d'Informatique de Nantes-Atlantique - UMR 6241
[1] Écoles des Mines de Nantes - 4, rue Alfred Kastler 44000 NANTES
[2] Université de Nantes - 2, rue de la Houssinière 44000 NANTES
[1]{prenom.nom}@mines-nantes.fr - [2]{prenom.nom}@univ-nantes.fr

Résumé

La programmation par contraintes s'attaque en général à des problèmes statiques, sans notion de temps. Cependant, les méthodes de réduction de domaines pourraient par exemple être utiles dans des problèmes portant sur des flux. C'est le cas de la vérification de programmes temps-réel, avec des variables dont les valeurs peuvent changer à chaque pas de temps. Dans cet article, nous nous intéressons à la vérification de domaines de variables (flux) dans le cadre d'un langage de diagrammes de blocs. Nous proposons une méthode de réduction de domaines de ces flux, pour encadrer finement les valeurs prises au cours du temps. En particulier, nous proposons un nouvel algorithme pour calculer un point fixe dans le cas des boucles temporelles. Nous présentons ensuite une application au langage FAUST, un langage fonctionnel temps réel pour le traitement audio et nous testons notre approche sur différents programmes FAUST standards.

Abstract

Constraint programming usually deals with static problems. However, domain reduction method could be useful in stream-based problems. This is the case in formal verification of real time programs for which variables can be assigned different values at every single time. In this paper, we focus on domain checking of stream variables in the context of block diagram languages. We propose a reduction algorithm for streams in order to tightly reduce their domains all over the time. Particularly, we propose a new technique to compute fix points of temporal loops. Finally, we apply our method to the FAUST language, which is a real time language for processing and generating audio streams. We also test some standards FAUST programs.

1 Introduction

La programmation par contraintes [10] offre un ensemble de méthodes efficaces pour modéliser et résoudre des problèmes combinatoires, mais pas seulement. Les méthodes de propagation de contraintes, notamment continues [1], peuvent approximer rapidement des ensembles définis dans un langage générique. Cela permet de les croiser naturellement avec les outils d'interprétation abstraite [4], où l'on souhaite vérifier un programme en calculant des sur-approximations de ses traces, comme cela a été fait dans différents travaux récents [7, 15, 14].

On s'intéresse dans cet article à un cas particulier de programmes, des DSP (Digital Signal Process), qui opèrent sur des flux infinis, et en particulier au langage FAUST (Functional Audio Stream) destiné à la synthèse et au traitement de flux audio [13]. Il est important de vérifier les programmes FAUST car le langage est destiné à des non-informaticiens et utilisé dans des concerts, cas où une erreur est inenvisageable. Plus précisément, nous souhaitons encadrer les valeurs prises par les flux calculés, notamment pour éviter des saturations sur les sons produits [1]. Avoir un encadrement des flux calculés permet aussi de pouvoir estimer la mémoire nécessaire au programme lorsqu'il contient des tables pour stocker certaines valeurs.

Un langage de flux comme FAUST contient des boucles, qui consistent à itérer certaines opérations à chaque pas de temps. Traitant de flux infinis, ces boucles n'ont pas de condition d'arrêt et sont toujours

1. une saturation se produit quand un son sort sort de $[-1, 1]$, la limite des sons numériques. Dans ce cas, il est en général coupé pour les valeurs au delà des bornes, ce qui modifie la forme de l'onde sonore et provoque un effet clairement audible.

infinies, ce qui ne facilite pas l'analyse des flux générés. Le compilateur de FAUST embarque un analyseur dans le domaine abstrait des intervalles [5, 6], mais l'opérateur d'élargissement des intervalles, appliqué aux boucles FAUST, renvoie presque toujours un domaine infini, qui ne permet pas de conclure.

Dans cet article, nous utilisons des méthodes de programmation par contraintes pour vérifier des programmes temps-réel travaillant sur des flux. Pour cela, nous proposons une modélisation en contraintes de diagrammes de blocs, qui décrivent ces programmes. La propagation des contraintes ainsi générées permet de calculer des sur-approximations des ensembles de valeurs prises par les flux au cours du temps. Nous introduisons une contrainte spécifique pour traiter des décalages temporels et proposons un algorithme de propagation efficace pour les problèmes incluant cette contrainte. Enfin, nous appliquons les méthodes développées au langage FAUST, un langage temps-réel de traitement du son. Les contraintes sont calculées dans la librairie de contraintes continues Ibex [2]. Nous présentons des tests préliminaires, avec de très bons temps de résolution, y compris pour des problèmes avec des décalages temporels.

Ce travail fait suite à [3], où une première modélisation partielle (uniquement pour certains composants de FAUST) a été proposée. Cet article présente une modélisation plus générale des langages de flux, ainsi qu'un nouvel algorithme efficace pour les boucles. Des travaux antérieurs comme [9, 8] proposent déjà de résoudre des contraintes sur des flux, mais il s'agit de calculer un automate dont les chemins sont les solutions des contraintes, ce qui diffère assez nettement de notre approche où le problème est d'utiliser la propagation de contraintes pour déterminer des propriétés sur les flux.

Cet article est organisé comme suit : la section 2 introduit la notion de diagrammes de blocs. La section 3 donne la modélisation des diagrammes de blocs en contraintes, dont l'algorithme de résolution est donné dans la section 4. Enfin, la section 5 détaille l'application visée et les tests réalisés.

2 Diagrammes de blocs

Dans cette section, nous présentons la notion de diagrammes de blocs qui sert à définir la sémantique des langages considérés.

2.1 Syntaxe

Un bloc est une fonction, appliquant un opérateur à un ensemble d'entrées ordonnées et produisant une ou plusieurs sorties ordonnées.

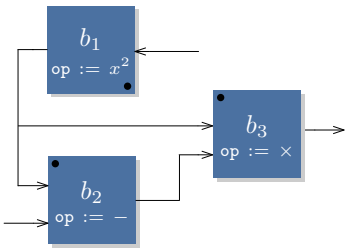


FIGURE 1 – Un diagramme de blocs de $\texttt{DBloc}(\mathbb{R})$

Definition 2.1 (Bloc) Soit E un ensemble non vide. Un bloc sur E est un triplet $b = (\texttt{op}, n, m)$ tel que :
- $n \in \mathbb{N}$ est appelé *nombre d'entrées* ;
- $m \in \mathbb{N}$ est appelé *nombre de sorties* ;
- $\texttt{op} : E^n \to E^m$ est appelé *opérateur*.

De plus, les n *entrées* et les m *sorties* sont ordonnées :
- $[i]b$ correspond à la $i^{\text{ème}}$ entrée $(1 \leq i \leq n)$;
- $b[j]$ correspond à la $j^{\text{ème}}$ sortie $(1 \leq j \leq m)$.

Pour tout bloc, nous écrirons que l'entrée i (resp. sortie j) existe ssi i est un entier compris entre 1 et le nombre d'entrées de ce bloc (resp. j, sorties). Dans la suite, étant donné un ensemble E non vide, $\texttt{Bloc}(E)$ correspond à l'ensemble des blocs sur E.

Definition 2.2 (Connecteur) Soit B un ensemble de blocs. Un connecteur défini sur B est un couple $(b[i], [j]b')$ tel que :
- b et b' sont des blocs de B ;
- la sortie i du bloc b existe ;
- l'entrée j du bloc b' existe.

Definition 2.3 (Diagramme de blocs) Soit E un ensemble non vide. Un diagramme de blocs sur E est un couple $d = (B, C)$ tel que :
- B est un ensemble de blocs sur E ;
- C est un ensemble de connecteurs sur B.

De la même manière que pour les blocs nous utilisons la notation $\texttt{DBloc}(E)$ pour l'ensemble des diagrammes de blocs sur E.

La Figure 1 représente trois blocs manipulant des réels (*i.e.* appartenant à $\texttt{Bloc}(\mathbb{R})$). Il s'agit du bloc b_1 muni de la fonction carrée comme opérateur ; du bloc b_2 avec la soustraction et du bloc b_3 avec la multiplication. L'ordre des entrées et sorties est indiqué par la distance de chacun au point noir. Quant aux connecteurs, ils sont matérialisés par des flèches allant d'un bloc à un autre. Ce diagramme de blocs en possède trois : le connecteur $(b_1[1], [1]b_2)$ allant de b_1 à b_2 ; $(b_1[1], [1]b_3)$ allant de b_1 à b_3 et $(b_2[1], [2]b_3)$ de b_2 à b_3. Le tout (ces trois blocs et ces trois connecteurs) forme un diagramme de blocs manipulant des réels (*i.e.* un diagramme de $\texttt{DBloc}(\mathbb{R})$).

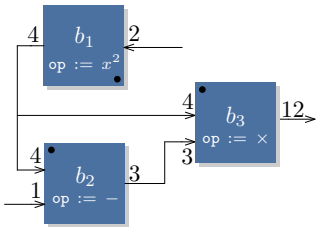


FIGURE 2 – Un diagramme de $\texttt{DBloc}(\mathbb{R})$ interprété

2.2 Sémantique

Après avoir défini ce que sont les diagrammes de blocs, nous définissons leur sémantique.

Definition 2.4 (Interprétation d'un bloc) Soit E un ensemble non vide. Une interprétation I d'un bloc $b = (\texttt{op}, n, m) \in \texttt{Bloc}(E)$ est donnée par l'association de chaque entrée i à un élément de E noté $I([i]b)$ et de chaque sortie j à un élément de E noté $I(b[j])$.

Definition 2.5 (Interprétation d'un diagramme de blocs) Soit E un ensemble non vide. Une interprétation I d'un diagramme de blocs $d = (B, C) \in \texttt{DBloc}(E)$ est telle que I est une interprétation pour chaque bloc b de B.

En reprenant le diagramme de $\texttt{DBloc}(\mathbb{R})$ en exemple dans la Figure 1, donner une interprétation à ce diagramme revient à poser un réel devant chaque entrée et devant chaque sortie des blocs (*e.g.* Figure 2 où nous lisons que $I([1]b_2)$ vaut 4 et que $I(b_3[1])$ vaut 12).

Definition 2.6 (Modèle d'un bloc) Soient E un ensemble non vide, $b = (\texttt{op}, n, m)$ un bloc de $\texttt{Bloc}(E)$ et I une interprétation de b.

$$I \text{ est un modèle de } b \Leftrightarrow$$
$$\texttt{op}(I([0]b), \ldots, I([n]b)) = (I(b[0]), \ldots I(b[m]))$$

Definition 2.7 (Modèle d'un diagramme de blocs) Soient E un ensemble non vide, $d = (B, C)$ un diagramme de blocs de $\texttt{DBloc}(E)$ et I une interprétation de d.

$$I \text{ est un modèle de } d \Leftrightarrow$$
$$\forall b \in B : I \text{ est un modèle de } b$$
$$\text{et } \forall (b[i], [j]b') \in C : I(b[i]) = I([j]b')$$

La Figure 2 propose un modèle au diagramme de blocs de la Figure 1. Remarquons qu'il s'agit d'un modèle parmi l'infinité de ceux que le diagramme admet.

Pour I une interprétation et d un diagramme de blocs, nous noterons $I \models d$ la relation : I est un modèle de d.

2.3 Cas des flux

Jusqu'ici, nous avons construit des diagrammes de blocs sur des ensembles quelconques. Dans la suite, nous prenons comme ensemble de base E un ensemble de *flux*. Pour cela, nous considérons que le temps est discrétisé et nous nous intéressons à des suites de valeurs pouvant changer à chaque pas de temps.

Definition 2.8 (Flux) Soit D un ensemble non vide. Un flux x sur D (appelé aussi un flux de domaine D) est une suite infinie à valeurs dans D.

Dans ce qui suit, nous appelons $x(t)$ la valeur au temps t du flux x ($t \in \mathbb{N}$). Pour un ensemble D non vide, nous notons $\mathbb{S}(D)$ l'ensemble des flux de domaine D. Parmi les blocs sur les flux, nous distinguons deux catégories.

Definition 2.9 (Bloc fonctionnel) Soit D un ensemble non vide et $b = (\texttt{op}, n, m) \in \texttt{Bloc}(\mathbb{S}(D))$.

$$b \text{ est fonctionnel } \Leftrightarrow$$
$$\exists f : D^n \to D^m \text{ telle que}$$
$$\forall s_1, \ldots, s_n, s'_1, \ldots, s'_m \in \mathbb{S}(D)$$
$$\text{avec } \texttt{op}(s_1, \ldots, s_n) = (s'_1, \ldots, s'_m) :$$
$$\forall t \in \mathbb{N} : f(s_1(t), \ldots, s_n(t)) = (s'_1(t), \ldots, s'_m(t))$$

Un bloc fonctionnel peut être calculé à chaque pas de temps indépendamment. Les blocs non fonctionnels ont des dépendances temporelles.

Definition 2.10 (Bloc temporel) Soit D un ensemble non vide et $b = (\texttt{op}, n, m) \in \texttt{Bloc}(\mathbb{S}(D))$.

$$b \text{ est temporel } \Leftrightarrow b \text{ n'est pas fonctionnel}$$

Parmi tous les blocs temporels, nous en identifions un particulier : le bloc *fby* (*followed by*).

Definition 2.11 (Bloc *fby*) Soit D un ensemble non vide. Le bloc $fby = (\texttt{op}, 2, 1)$ de $\texttt{Bloc}(\mathbb{S}(D))$ est défini comme suit :

$$\texttt{op} : \mathbb{S}(D) \times \mathbb{S}(D) \to \mathbb{S}(D)$$
$$(s_1, s_2) = s_3$$

$$\text{avec } s_3(t) = \begin{cases} s_1(0) & \text{si } t = 0 \\ s_2(t-1) & \text{sinon} \end{cases}$$

Remarque Un diagramme de blocs sur des flux peut contenir des cycles (équivalent d'une boucle en programmation). En pratique, si ces boucles ne contiennent pas de *fby*, elles provoquent un calcul infini à chaque pas de temps sauf dans des cas très particuliers (fonctions nulles par exemple). Mais dès lors qu'elles contiennent au moins un bloc *fby* dans le cycle, ce phénomène ne se produit pas.

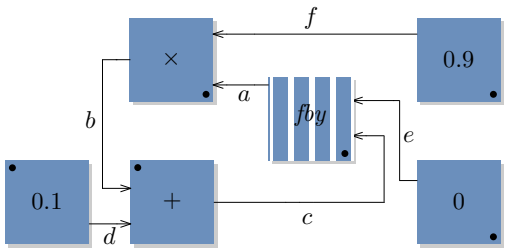


FIGURE 3 – Un diagramme de blocs de $\mathtt{DBloc}(\mathbb{S}(\mathbb{R}))$

t	0	1	2	3	4	...
$a(t)$	0	0.1	0.19	0.27	0.34	...
$b(t)$	0	0.09	0.17	0.24	0.31	...
$c(t)$	0.1	0.19	0.27	0.34	0.41	...
$d(t)$	0.1	0.1	0.1	0.1	0.1	...
$e(t)$	0	0	0	0	0	...
$f(t)$	0.9	0.9	0.9	0.9	0.9	...

TABLE 1 – Modèle du diagramme de la Figure 3 pour les 5 premiers temps.

Dans toute la suite, on supposera que les diagrammes de blocs considérés contiennent toujours au moins un *fby* dans chacun de leurs cycles.

La Figure 3 illustre un diagramme de blocs qui, entre autres, possède un cycle contenant un bloc *fby*. [2] Dans ce diagramme, les blocs 0, 0.1 et 0.9 correspondent aux opérateurs constants (*i.e.* $\forall t \in \mathbb{N} : 0.9(t) = 0.9$). La Table 1 propose un modèle pour les 5 premiers temps. Nous pouvons par exemple y lire le décalage d'un temps entre a et c dû au bloc *fby*.

3 Vérification des flux par contraintes

Les diagrammes de blocs sur des flux peuvent représenter des programmes. Dans ce cas, il est intéressant de connaître certaines propriétés sur les flux générés, qu'il s'agisse des sorties du programme ou des flux intermédiaires (variables locales). Nous présentons ici une modélisation, sous la forme d'un problème de contraintes, du problème suivant : encadrer les valeurs prises par les flux d'un diagramme de blocs.

3.1 Abstraction du temps

La première étape de notre modélisation consiste à abstraire le temps, de façon à considérer les enveloppes des flux générés.

Definition 3.1 (Abstraction temporelle) L'abstraction temporelle d'un flux x est notée $\dot{x}$ et vaut l'ensemble des valeurs prises par ce flux :

$$\dot{x} = \bigcup_{t \in \mathbb{N}} x(t)$$

Il est important de souligner que les abstractions temporelles peuvent contenir un nombre infini d'éléments. De fait, une représentation en extension n'est pas envisageable. Nous nous intéresseront dans la suite à des sur-approximations des flux dans les intervalles.

2. Dans la représentation graphique des diagrammes de blocs sur des flux, nous hachurons les blocs temporels pour les différencier des blocs fonctionnels.

3.2 Abstraction en intervalles

Dans ce qui suit, nous faisons l'hypothèse que D est un ensemble totalement ordonné et nous notons $\mathbb{I}_D$ l'ensemble des intervalles à valeurs dans D. Pour tout intervalle I, nous notons $\lceil I \rceil$ sa borne supérieure et $\lfloor I \rfloor$ sa borne inférieure.

Definition 3.2 (Sur-approximation) Soit D un ensemble non vide muni d'un ordre total $\leq$ et S un ensemble inclus dans D. Une sur-approximation dans les intervalles de S est un intervalle englobant noté $[S]$ appartenant à $\mathbb{I}_{\overline{D}}$.

$$[S] = \{a \leq x \leq b \mid \forall s \in S : a \leq s \leq b\}$$
avec a, b et x dans $\overline{D}$ [3]

Definition 3.3 (Sur-approximation minimale) Soit D un ensemble non vide muni d'un ordre total $\leq$ et S un ensemble inclus dans D. La sur-approximation minimale de S notée $[\![S]\!]$ est la plus petite sur-approximation pour l'inclusion ensembliste.

Proposition 3.1 (Treillis des sur-approximations) Soit D un ensemble non vide muni d'un ordre total $\leq$ et S un ensemble inclus dans D. L'ensemble des sur-approximations $[S]$ muni de l'inclusion comme relation d'ordre forme un treillis dont la borne supérieure est l'union ensembliste et la borne inférieure l'intersection ensembliste. Le plus petit élément correspond à la sur-approximation minimale $[\![S]\!]$ et le plus grand à l'ensemble étendu $\overline{D}$.

Corollaire 3.2 Soit D un ensemble non vide muni d'un ordre total $\leq$. Pour tout S inclu dans D, la sur-approximation minimale existe et est unique.

La démonstration est évidente et laissée au lecteur.

3.3 Modélisation par contraintes

Un diagramme de blocs calcule des sorties en fonction de ses entrées. Pour calculer une sur-approximation des flux générés par un diagramme de

3. $\overline{D}$ appelé ensemble étendu de D correspond à l'union de D et de ses limites (*e.g.* $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$)

$$
\begin{array}{lll}
\mathrm{a} & := & \mathit{fby}(\mathrm{e}\,,\ \mathrm{c}) \\
\mathrm{b} & := & \times(\mathrm{f}\,,\mathrm{a}) \\
\mathrm{c} & := & +(\mathrm{b}\,,\mathrm{d})
\end{array}
\qquad
\begin{array}{lll}
\mathrm{d} & := & 0.1 \\
\mathrm{e} & := & 0 \\
\mathrm{f} & := & 0.9
\end{array}
$$

FIGURE 4 – Modèle en contraintes sur les flux

$$
\begin{array}{lll}
\mathrm{a} & := & [\mathit{fby}](\mathrm{e}\,,\ \mathrm{c}) \\
\mathrm{b} & := & [\times](\mathrm{f}\,,\mathrm{a}) \\
\mathrm{c} & := & [+](\mathrm{b}\,,\mathrm{d})
\end{array}
\qquad
\begin{array}{lll}
\mathrm{d} & := & [0.1] \\
\mathrm{e} & := & [0] \\
\mathrm{f} & := & [0.9]
\end{array}
$$

FIGURE 5 – Modèle en contraintes sur les intervalles

bloc (B, C) de $\texttt{DBloc}(\mathbb{S}(D))$, nous associons à chaque entrée et à chaque sortie des blocs de B une *variable* de domaine $\mathbb{S}(D)$ et nous considérons l'opérateur de chaque bloc comme une *contrainte* liant les sorties aux entrées. Par exemple, l'opérateur + du diagramme de la Figure 3 calcule c en fonction de b et de d, de sorte que : $c = b + d$. De plus, nous ajoutons une contrainte d'égalité pour les variables qui composent chaque connecteur de C. Dans notre exemple, nous avons unifié les variables ainsi connectées (*e.g.* $[1]+ = *[1] = \mathrm{b}$). La Figure 4 présente un modèle en contraintes sur les flux de l'exemple de la Figure 3.

Nous supposons donc que chaque opérateur du diagramme de blocs peut être traduit dans le langage de contraintes. A partir de ce modèle de contraintes sur les flux, nous considérons le modèle sur les intervalles contenant exactement les mêmes formulations de contraintes que celui sur les flux, en remplaçant chaque fonction par son extension aux intervalles comme défini dans [11]. Dans ce modèle, le domaine des variables n'est plus $\mathbb{S}(D)$ mais $\mathbb{I}_D$.

Definition 3.4 (Extension aux intervalles) Soit D un ensemble non vide et $f : \mathbb{S}(D)^n \to \mathbb{S}(D)^m$ une fonction $(n, m \in \mathbb{N})$. Une extension aux intervalles de f, notée $[f]$ est telle que :

$$
\begin{array}{rcl}
[f] \ : \quad (\mathbb{I}_D)^n & \to & (\mathbb{I}_D)^m \\
X_1, \ldots, X_n & \mapsto & Y_1, \ldots, Y_m \quad \text{avec} \\
Y_i = [\{\dot{y_i} \mid x_j \in X_j, y_1, \ldots, y_m = f(x_1, \ldots, x_n)\}] &&
\end{array}
$$

La Figure 5 présente un modèle en contraintes sur les intervalles de l'exemple de la Figure 4.

En propageant les contraintes pour le modèle sur les flux, avec la bonne définition des fonctions étendues aux intervalles pour le modèle sur les intervalles, nous pouvons calculer des sur-approximations pour chaque flux. Ainsi, nous avons traduit un diagramme de blocs en un problème de contraintes, dans un langage *ad hoc*, tel qu'en propageant les contraintes, on obtient un encadrement (sur-approximation) des valeurs prises par les flux du diagramme de bloc.

Variable	a	b	c	d	e	f
Solution	$[0; 1]$	$[0; 0.9]$	$[0; 1]$	$[0.1]$	$[0]$	$[0.9]$

TABLE 2 – Solutions au modèle de contraintes sur les intervalles de la Figure 5

Le Tableau 2 présente la solution minimale au modèle de contrainte sur intervalles de la Figure 5.

4 Résolution

Dans cet article, nous résolvons uniquement les diagrammes de blocs utilisant exclusivement le bloc *fby* comme bloc temporel. Cette restriction n'est pas trop forte puisque ce bloc suffit à exprimer tous les diagrammes de blocs avec une mémoire finie.

4.1 Graphe des dépendances

Pour résoudre les problèmes de contraintes issus des diagrammes de blocs, nous utilisons les dépendances entre les flux, devenus des variables du CSP.

Definition 4.1 (Graphe des dépendances) Soient E un ensemble non vide et $d = (B, C)$ un diagramme de blocs de $\texttt{DBloc}(E)$. La relation de dépendances notée $\to$ est la relation sur l'ensemble des entrées et sorties des blocs de B contenant exactement les couples suivants :

$$\forall (b[i], [j]b') \in C : b[i] \to [j]b'$$
$$\forall b = (\texttt{op}, n, m) \in B : [i]b \to b[j]$$
$$(1 \leq i \leq n) \text{ et } (1 \leq j \leq m)$$

Du point de vue de la programmation par contraintes, le graphe dessiné par cette relation est le graphe de dépendance des contraintes (dont les nœuds sont les variables du problème), mais dont les arêtes sont orientées par la dépendance induite par les blocs/-contraintes. La Figure 6 correspond au graphe des dépendances de l'exemple de la Figure 3. [4]

Le calcul des composantes fortement connexes sur ce graphe a un intérêt double. Premièrement, il permet de découper le problème global en sous-problèmes, ces sous-problèmes pouvant être traités dans l'ordre des arêtes qui les relient. Deuxièmement, l'algorithme standard de [16] retourne les composantes dans l'ordre de leurs dépendances dans le graphe orienté, ce que nous utilisons pour choisir l'ordre de la propagation des contraintes (*e.g.* pour la Figure 6, nous traiterons d'abord les composantes du pourtour avant de s'intéresser à celle du centre).

4. Comme dans la représentation graphique des diagrammes de blocs, nous distinguons les blocs temporels des blocs fonctionnels en hachurant cette fois-ci les flèches des blocs temporels.

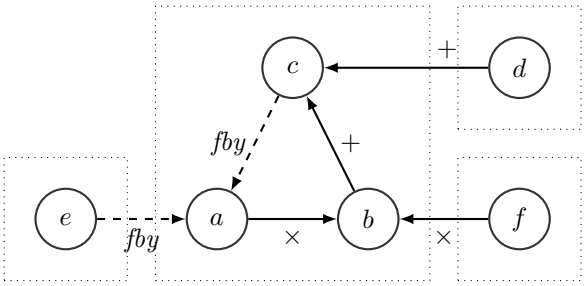


FIGURE 6 – Dépendances inter-connecteurs

Remarque Les cycles du graphe des dépendances sont produits par les boucles du diagramme de blocs.

4.2 Fonction de transfert

L'algorithme 1 propose une méthode de résolution des composantes fortements connexes du graphe des dépendances des contraintes. Nous rappelons l'hypothèse de la section 2.3 : chaque cycle du diagramme de blocs contient au moins un bloc *fby*.

Lorsqu'un bloc *fby* appartient à un cycle, cela implique que son *entrée* dépend de sa *sortie* et donc a fortiori que le flux de *sortie* au temps t dépend de sa valeur au temps précédent (cf. Définition 2.11 introduisant la sémantique du bloc *fby*). Un tel flux de sortie x peut alors se définir récursivement comme suit :

$$\forall t \in \mathbb{N} : x(t+1) = f(x(t))$$
$$\text{et } x(0) \text{ vaut la valeur au temps 0 de } [0]fby$$

Pour trouver cette fonction, appelée *fonction de transfert de la boucle*, nous partons du diagramme de blocs générant la composante fortement connexe considérée et nous retirons les connecteurs amenant à des blocs *fby* (lignes 10 à 17 de l'Algorithme 1). De ce nouveau diagramme, un parcours en profondeur du dual de son graphe des dépendances donne exactement la fonction de transfert de la boucle en notation polonaise inversée. [5]

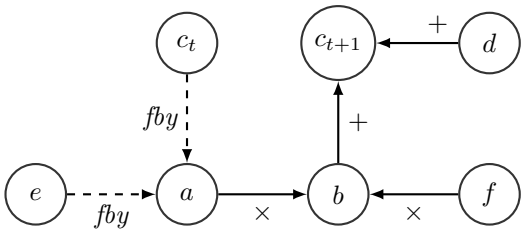


FIGURE 7 – Graphe d'une fonction de transfert

5. Il est nécessaire de respecter l'ordre des entrées imposées par le diagramme de blocs pour la selection des fils (tous les opérateurs ne sont pas forcément commutatifs).

```
1:  fonction RÉSOLUTIONCOMPOSANTE(
2:          C : Set<Contrainte>,
3:          domaine : Map<Variable, I_D>)
4:      retourne Map<Variable, I_D>
5:
6:      // Retrait des connexions vers fby
7:      entrees : List<Variable>
8:      sorties : List<Variable>
9:      compteur ← 0 : entier
10:     pour chaque contrainte b[i] = [j]b' dans C faire
11:         si b' égale fby alors
12:             C ← C \ {b[i] = [j]b'}
13:             entrees.AJOUTER([j]b')
14:             sorties.AJOUTER(b[i])
15:             compteur ← compteur + 1
16:         fin si
17:     fin pour chaque
18:
19:     // Fonction de transfert
20:     f : (I_D)^compteur → (I_D)^compteur
21:     f ← FONCTIONTRANSFERT(C, domaine,
22:                           entrees, sorties)
23:     approximation : Liste<I_D>
24:     approximation ← SURAPPROXIMATION(f)
25:
26:     //  MAJ domaines des variables
27:     pour chaque i de 1 à compteur faire
28:         domaine[entrees[i]] ← approximation[i]
29:         domaine[sorties[i]] ← approximation[i]
30:     fin pour chaque
31:     PROPAGER(C, domaine, entrees ∪ sorties)
32:
33:     retourne domaine
34: fin fonction
```

Algorithm 1 – Résolution des composantes fortement connexes

La Figure 7 représente le graphe des dépendances du diagrammes de blocs de la composante centrale de la Figure 6 où le connecteur menant au bloc *fby* a été supprimé. Par lecture de ce graphe, la fonction de transfert de la boucle en notation infixée est $c_{t+1} = f \times fby(e, c_t) + d$.

Ensuite, en utilisant les extensions aux intervalles des opérateurs des blocs (cf. Définition 3.4) et en remplaçant les variables n'appartenant pas à la composante connexe par leur domaine, nous obtenons la fonctions de transfert sur les intervalles de la boucle (dans l'Algorithme 1, cette fonction est récupérée via l'appel à FONCTIONTRANSFERT de la ligne 21). Dans notre exemple, elle vaut $F(X) = [0.9] \times [fby]([0], X) + [0.1]$ avec $X \in \mathbb{I}_{\mathbb{R}}$.

La fonction de transfert sur les intervalles et la clé de voûte pour la résolution de notre problème de part la Proposition 4.1.

```
fonction surApproximation(f : D^n → D^n)
    retourne Liste<I_D>

    etat : Liste<String>
    courant, min, max, image : Liste<I_D>
    pour chaque i de 1 à n faire
        etat[i] ← "Widening"
        courant[i] ← ∅
        min[i] ← ∅
        max[i] ← D
    fin pour chaque

    tant que continuerBoucle(courant, min, max)
    faire
        image ← f(courant)
        pour chaque i de 1 à n faire
            si etat[i] = "Widening" et
                    image[i] ⊆ courant[i] alors
                etat[i] ← "Narrowing"
                max[i] ← courant[i]
            sinon si etat[i] = "Narrowing" et
                    image[i] ⊈ courant[i] alors
                etat[i] ← "Widening"
                min[i] ← courant[i]
            fin si

            courant[i] = courant[i] ∪ image[i]
            si etat[i] = "Widening" alors
                courant[i] ←
          selectIntervalBetween(courant[i], max[i])
            sinon
                courant[i] ←
          selectIntervalBetween(min[i], courant[i])
            fin si
        fin pour chaque
    fin tant que
    retourne max
fin fonction
```

Algorithm 2 – Fonction de recherche aléatoire

Proposition 4.1 Soient D un ensemble non vide et $d = (B, C)$ un diagramme de blocs de $\texttt{DBloc}(\mathbb{S}(D))$. Pour tout bloc *fby* $b \in B$ appartenant à une boucle de d dont la fonction de transfert sur les intervalles est F, si un intervalle S de $\mathbb{I}_D$ est stable par F, alors S est une sur-approximation valide des flux en sortie de b.

L'Algorithme 2 propose une méthode héritée de l'interprétation abstraite pour déterminer des ensembles stables par la fonction de transfert sur les intervalles. Cette fonction ne prétend pas retourner systématiquement la sur-approximation minimale mais simplement une sur-approximation acceptable.

Pour chaque variable d'entrée i de la fonction de transfert nous associons un espace de recherche délimité par les intervalles $min[i]$ et $max[i]$ qui sont respectivement initialisés à l'ensemble vide et à l'ensemble étendu du domaine considéré. A chaque tour de boucle, $courant[i]$ est sélectionné tel qu'il contienne $min[i]$ et soit inclus dans $max[i]$ (*i.e.* $min[i] \subseteq courant[i] \subseteq max[i]$ est un invariant de boucle).

```
 1: fonction réductionDomaines((B, C) : DBloc(S(D)))
 2:     retourne Map<Variable, I_D>
 3:
 4:     // Initialisation
 5:     domaine : Map<Variable, I_D>
 6:     b : Bloc(S(D))
 7:     pour chaque b ∈ B faire
 8:         pour chaque entrée [i]b de b faire
 9:             domaine[[i]b] ← [D]
10:         pour chaque sortie b[j] de b faire
11:             domaine[b[j]] ← [D]
12:     fin pour chaque
13:
14:     // Pré-traitement
15:     graphe : GrapheOrienté<Contrainte>
16:     graphe ← grapheDépendances(B, C)
17:     composante : Set<Connecteur>
18:     composantes : Pile<Set<Contrainte>>
19:     composantes ← composantesFortConnexes(
20:                                         graphe)
21:
22:     // Corps de la fonction
23:     tant que composantes non vide faire
24:         composante ← dépiler(composantes)
25:         domaine ← résolutionComposante(
26:                             composante, domaine)
27:     fin tant que
28:
29:     retourne domaine
30: fin fonction
```

Algorithm 3 – Algorithme global

Nous associons également à chaque variable d'entrée i une variable d'état $state[i]$ qui a pour valeur "Widening" ou "Narrowing". Il y a passage du *widening* au *narrowing* lorsque l'intervalle $courant[i]$ est stable par la fonction de transfert et passage du *narrowing* au *widening* dans le cas contraire.

Concernant la fonction selectIntervalBetween, la méthode de sélection est libre mais nous expliciterons dans nos expérimentations les choix réalisés (*i.e.* aléatoire, dichotomiques, etc.). De même pour la fonction continuerBoucle qui généralement possèdera un compteur pour limiter le nombre d'itérations. Dans les expérimentations, nous appellons selectIntervalBetween l'heuristique de selections et continuerBoucle l'heuristique de fin de boucle.

Finalement, l'Algorithme 3 correspond à la résolution complète du problème.

5 Faust

Nous appliquons ici les outils développés au langage FAUST, un langage développé par le Grame [6].

5.1 Présentation

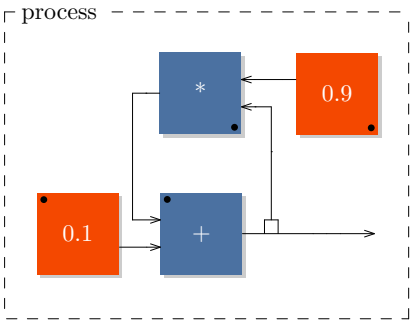


FIGURE 8 – Un diagramme de blocs généré par FAUST

FAUST (Functional Audio Stream) est un langage fonctionnel pour la synthèse et le traitement temps-réel de flux audio. L'une de ses particularités est d'avoir une sémantique bien définie, dans un langage de diagrammes de blocs [12]. Le compilateur FAUST peut générer pour chaque programme le diagramme de blocs qui en est la sémantique.

Un programme FAUST est un processeur de signal numérique (DSP : Digital Signal Processor). Le concept des DSP n'est pas propre à FAUST, ni au traitement audio : il est utilisé dans toutes les applications qui reçoivent et traitent des signaux numériques : modems, appareils multimédia, récepteurs GPS, systèmes vidéo, ... FAUST propose un langage complètement spécifié pour écrire les DSPs.

```
1 A = + (0.1);
2 B = * (0.9);
3 process = A ~ B;
```

FIGURE 9 – Exemple de DSP pour FAUST

Chaque DSP ainsi écrit passe dans le compilateur FAUST et produit un programme C++ optimisé, qui est ensuite compilé pour une plateforme visée. Ce processus de compilation en deux étapes permet à un même programme FAUST de pouvoir être compilé pour des téléphones, navigateurs web, dispositifs de concerts, etc. Nous nous intéressons ici à la première étape de compilation, qui permet de produire le code C++. En particulier, la première phase d'analyse syntaxique ré-écrit le programme sous une forme normale unique, dans laquelle sont notamment éliminées les formules redondantes (par exemple $x - x$). Cette forme normale nous permet de travailler sur des expressions avec peu d'occurences multiples pour les variables. Une fois cette étape réalisée, le compilateur calcule le diagramme de blocs résultant du programme. Le programme de la Figure 9 donne ainsi le diagramme présenté Figure 8 (copie d'écran de la sortie du compilateur FAUST).

6. `http://faust.grame.fr`

Diagramme FAUST Sémantique FAUST

$$\begin{cases} c(0) & = a(0) \\ c(t+1) & = b(t), \text{ si } t > 0 \end{cases}$$

Diagramme de blocs équivalent

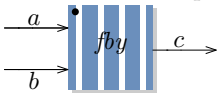
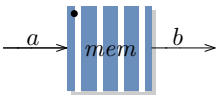


FIGURE 10 – Bloc *prefix* de FAUST

Diagramme FAUST Sémantique FAUST



$$\begin{cases} b(0) & = 0 \\ b(t+1) & = a(t), \text{ si } t > 0 \end{cases}$$

Diagramme de blocs équivalent

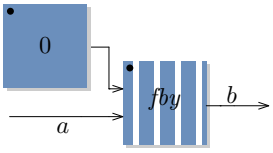


FIGURE 11 – Bloc *mem* de FAUST

5.2 Modélisation

Notre algorithme ne peut résoudre que des diagrammes de blocs de flux utilisant exclusivement le bloc *fby* comme bloc temporel. Or, il existe dans le langage des diagrammes de blocs de FAUST trois blocs temporels : *prefix*, *mem* et *delay*. Les figures 10, 11 et 12 rappellent la sémantique de ces blocs et proposent des diagrammes de blocs équivalents (possédant exactement les mêmes modèles) utilisant le bloc *fby*.

En utilisant l'équivalence proposée du bloc *delay* à la Figure 12, la complexité spatiale du pré-traitement peut devenir importante. Cependant, pour notre problème de sur-approximation des flux, le fait que ces derniers soient infinis alors que le nombre de *fby* ajoutés ne l'est pas, nous permet d'écrire un seul bloc *fby* pour avoir des solutions équivalentes (*i.e.* de par l'abstraction temporelle de la Définition 3.1). Dès lors, nous

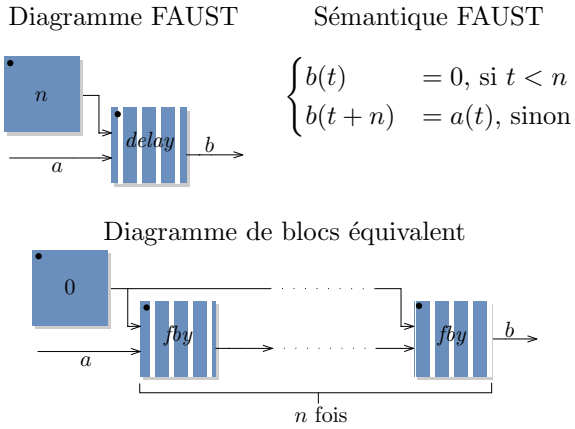


FIGURE 12 – Bloc *delay* de FAUST

utilisons la réécriture du bloc *mem* de la Figure 11 pour le bloc *delay*.

Une extension aux intervalles de l'opérateur *fby* en accord avec sa sémantique des équivalences proposées pour les blocs temporels est la suivante : $[\![fby]\!](A, B) = [A \cup B]$. Le reste du langage FAUST est principalement composé des fonctions arithmétiques usuelles, de fonctions importées de C++ (dont *sin*; *cos*, *exp*...) et d'opérateurs de comparaison. [7] Tous ces opérateurs s'étendent aux intervalles à la façon de [11], et la traduction par contraintes se fait naturellement dans un langage usuel de contraintes continues, excepté les opérateurs bit à bit.

5.3 Expériences

Avant de passer aux expérimentations, nous introduisons la notion de distance sur les intervalles afin de pouvoir mesurer la qualité des solutions retournées par notre algorithme.

Definition 5.1 (Distance étendue) Soit D un ensemble non-vide. La fonction de distance étendue $\overline{d}$ pour deux éléments x et y de $\overline{D}$ est donnée par :

$$\overline{d}(x, y) = 0 \qquad \text{si } x = y,$$
$$\overline{d}(x, y) = |x - y| \qquad \text{sinon.}$$

Cette définition permet d'évaluer la distance avec les infinis. Par exemple $\overline{d}(-\infty, 0)$ vaut $-\infty$ et $\overline{d}(+\infty, +\infty)$ retourne 0.

Definition 5.2 (Distance d'intervalles) Soit D un ensemble non-vide. La fonction de distance pour deux intervalles X et Y de $\mathbb{I}_{\overline{D}}$ est donnée par :

$$d(X, Y) = \overline{d}(\lfloor X \rfloor, \lfloor Y \rfloor) + \overline{d}(\lceil X \rceil, \lceil Y \rceil) \qquad \text{si } X \subseteq Y,$$
$$d(X, Y) = +\infty \qquad \text{sinon.}$$

7. voir `http://faust.grame.fr/index.php/documentation/references` pour une présentation complète.

Proposition 5.1 Soit un ensemble S inclus dans E, une sur-approximation $[S]$ est minimale ssi la distance entre $[S]$ et $[\![S]\!]$ est nulle.

Une bonne approximation est donc une approximation ayant la distance la plus petite possible à la sur-approximation minimale.

Nous avons testé notre approche sur cinq programmes FAUST : EXEMPLE-PAPIER correspond à l'exemple qui a été déroulé tout au long de ce papier ; COMPTEUR correspond à un flux commençant à 0 s'incrémentant de 1 à chaque temps (dont la borne supérieure est infinie) ; BRUIT-BLANC correspond à la génération d'un bruit (*i.e.* suite de valeurs aléatoires) ; SON-SINUS correspond à la génération d'un son pur et SON-ECHO à la création d'un écho simple sur une entrée audio.

Pour chaque exécution, l'heuristique de fin de boucle autorise un maximum de 5 branches de recherches avec pour chaque branche une limite de 500 selections d'intervalles (*i.e.* 500 tours de boucles). L'heuristique de selection des intervalles alterne entre trois : choisir aléatoirement une nouvelle borne inférieure entre $\lfloor min \rfloor$ et $\lfloor max \rfloor$, choisir aléatoirement une nouvelle borne supérieure entre $\lceil min \rceil$ et $\lceil max \rceil$, ou alors réaliser les deux en même temps.

La Figure 13 présente les résultats obtenus en appliquant notre algorithme de résolution pour une sélection de 5 programmes fondamentaux en FAUST. L'algorithme a été exécuté 10 fois pour chaque problème. Nous présentons à chaque fois les moyennes de temps d'exécution, de distance à la sur-approximation minimale et du nombre d'itérations.

Ces tests préliminaires, qui seront complétés par d'autres programmes de la librairie standard de FAUST, montrent que notre algorithme retourne toujours la plus petite sur-approximation des flux pour ce jeu d'essais. Le temps de calcul, qui doit être très rapide pour que notre outil puisse être embarqué dans le compilateur FAUST, est encourageant : le temps de calcul est de l'ordre d'une milli-seconde, et varie peu.

5.4 Conclusion

Nous avons présenté une modélisation en contraintes d'un problème de vérification pour des programmes temps-réel. Le but à terme est d'implémenter l'outil dans le compilateur FAUST, ce qui suppose que les temps de calcul soient très petits (négligeables devant le temps de compilation). Les premiers tests réalisés sont encourageant, et nous engagent à poursuivre le développement de l'outil. Dans un futur proche, ce travail sera implémenté de façon à pouvoir être intégré dans le compilateur FAUST, et les méthodes seront

Programme	#blocs (dont *fby*)	Temps (10^{-3}s)			Distance			#itérations		
		Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
EXEMPLE-PAPIER	6 (1)	0,828	0,8727	1,016	0	0	0	851	911,2	1176
COMPTEUR	4 (1)	1,174	1,5072	1,887	0	0	0	1646	2124	2500
BRUIT-BLANC	8 (1)	0,784	0,9888	1,568	0	0	0	1007	1007	1007
SON-SINUS	5 (1)	0,326	0,3906	0,899	0	0	0	13	13	13
SON-ECHO	7 (1)	0,134	0,1376	0,14	0	0	0	1	1	1

FIGURE 13 – Résultats pour la résolution du jeu d'essais

testées intensivement pour améliorer le réglage des paramètres.

Par ailleurs, ce travail montre l'intérêt de l'utilisation de techniques de programmation par contraintes dans des cadres exotiques, ici, sur des variables flux. L'algorithme développé étant générique, il pourrait avoir des utilisations à d'autres langages de signaux, ce que nous examinerons par la suite.

Références

[1] Frédéric Benhamou and William J. Older. Applying interval arithmetic to real, integer and boolean constraints. *Journal of Logic Programming*, 32(1) :1–24, 1997.

[2] Gilles Chabert and Luc Jaulin. Contractor programming. *Artificial Intelligence*, 173(11) :1079–1100, 2009.

[3] Yann Orlarey Charlotte Truchet, Julie Laniau. Avoiding saturation in sound processes with constraint programming. In *Constraint Programming meets Verification Workshop at CP'14*, 2014.

[4] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.

[5] P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977.

[6] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4) :511–547, 1992.

[7] Tristan Denmat, Arnaud Gotlieb, and Mireille Ducassé. An abstract interpretation based combinator for modeling while loops in constraint programming. In *In Proceedings of Principles and Practices of Constraint Programming (CP'07), Springer Verlag, LNCS 4741*, pages 241–255, 2007.

[8] Arnaud Lallouet, Yat Chiu Law, Jimmy HM Lee, and Charles FK Siu. Constraint programming on infinite data streams. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume One*, pages 597–604. AAAI Press, 2011.

[9] JasperC.H. Lee and JimmyH.M. Lee. Towards practical infinite stream constraint programming : Applications and implementation. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming*, volume 8656 of *Lecture Notes in Computer Science*, pages 449–464. Springer International Publishing, 2014.

[10] Ugo Montanari. Networks of constraints : Fundamental properties and applications to picture processing. *Information Science*, 7(2) :95–132, 1974.

[11] Ramon Edgar Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs N. J., 1966.

[12] Y. Orlarey, D. Fober, and S. Letz. An algebra for block diagram languages. In *International Computer Music Conference*, 2002.

[13] Y. Orlarey, D. Fober, and S. Letz. Syntactical and semantical aspects of faust. *Soft Computing*, 8(9) :623–632, 2004.

[14] Marie Pelleau, Charlotte Truchet, and Frédéric Benhamou. The octagon abstract domain for continuous constraints. *Constraints*, 19(3) :309–337, 2014.

[15] Olivier Ponsini, Claude Michel, and Michel Rueher. Refining abstract interpretation based value analysis with constraint programming techniques. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, pages 593–607, 2012.

[16] Micha Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1) :67–72, 1981.

Un algorithme incrémental dirigé par les flots et basé sur les contraintes pour l'aide à la localisation d'erreurs

Mohammed Bekkouche **Hélène Collavizza** **Michel Rueher**

Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France
{bekkouche,helen,rueher}@unice.fr

Résumé

Dans cet exposé, nous présentons notre algorithme amélioré [1] de localisation d'erreurs à partir de contre-exemples, LocFaults, basé sur la programmation par contraintes et dirigé par les flots. Cet algorithme analyse les chemins du CFG (Control Flow Graph) du programme erroné pour calculer les sous-ensembles d'instructions suspectes permettant de corriger le programme. En effet, nous générons un système de contraintes pour les chemins du graphe de flot de contrôle pour lesquels au plus k instructions conditionnelles peuvent être erronées. Ensuite, nous calculons les MCS (Minimal Correction Set) de taille limitée sur chacun de ces chemins. La suppression de l'un de ces ensembles de contraintes donne un sous-ensemble satisfiable maximal, en d'autres termes, un sous-ensemble maximal de contraintes satisfaisant la postcondition. Pour calculer les MCS, nous étendons l'algorithme générique proposé par Liffiton et Sakallah [11, 12] dans le but de traiter des programmes avec des instructions numériques plus efficacement. Nous nous intéressons à présenter l'aspect incrémental de ce nouvel algorithme qui n'est pas encore présenté aux JFPC.

Considérons le programme `AbsMinus` (voir fig. 1). Les entrées sont des entiers $\{i,\ j\}$ et la sortie attendue est la valeur absolue de $i-j$. Une erreur a été introduite sur la ligne 10, ainsi pour les données d'entrée $\{i = 0, j = 1\}$, `AbsMinus` retourne -1. La post-condition est juste $result = |i - j|$.

Le graphe de flot de contrôle (CFG) du programme `AbsMinus` et un chemin erroné sont représentés dans la figure 2. Ce chemin erroné correspondent aux données d'entrée : $\{i = 0, j = 1\}$. Tout d'abord, LocFaults collecte sur le chemin 2.(b) l'ensemble de contraintes $C_1 = \{i_0 = 0, j_0 = 1, k_0 = 0, k_1 = k_0 + 2, r_1 = i_0 - j_0\}$ [1].

1. Nous utilisons la transformation en forme DSA [5] qui assure que chaque variable est affectée une seule fois sur chaque chemin du CFG.

```
1 class AbsMinus {
2 /*Il renvoie |i−j|, la valeur absolue de i moins j*/
3 /*@ ensures
4  @ ((i < j) ==> (\result == j−i)) &&
5  @ ((i >= j) ==> (\result == i−j)); */
6  int AbsMinus (int i, int j) {
7     int result;
8     int k = 0;
9     if (i <= j) {
10          k = k+2; } // erreur: k = k + 2 au lieu de k = k + 1
11     if (k == 1 && i != j) {
12          result = j−i; }
13     else {
14          result = i−j; }
15     return result;  } }
```

FIGURE 1 – Le programme AbsMinus

Puis, LocFaults calcule les MCS de C_1. Seulement un MCS peut être trouvé dans C_1 : $\{r_1 = i_0 - j_0\}$. En d'autres termes, si nous supposons que les instructions conditionnelles sont correctes, la seule instruction suspecte sur ce chemin erroné est l'instruction 14.

Ensuite, LocFaults commence le processus de déviation. La première déviation (voir la figure 3.(a), chemin vert) produit encore un chemin qui viole la post-condition, et donc, nous l'ignorons. La second déviation (voir la figure 3.(b), chemin bleu) produit un chemin qui satisfait la postcondition. Donc, LocFaults collecte les contraintes sur la partie du chemin 3.(b) qui précède la condition déviée, c'est-à-dire $C_2 = \{i = 0, j = 1, k_0 = 0, k_1 = k_0 + 2\}$. Puis LocFaults recherche les MCS de $C_2 \cup \neg(k = 1 \wedge i \neq j)$; c'est-à-dire nous essayons d'identifier les instructions qui doivent être modifiées afin que le programme aura un chemin satisfaisant la post-condition pour les données d'entrée. Ainsi, pour cette deuxième déviation deux instructions suspectes sont identifiées :

- L'instruction conditionnelle sur la ligne 11 ;
- L'affectation sur la ligne 10 car la contrainte correspondante est le seul MCS dans $C_2 \cup \neg(k =$

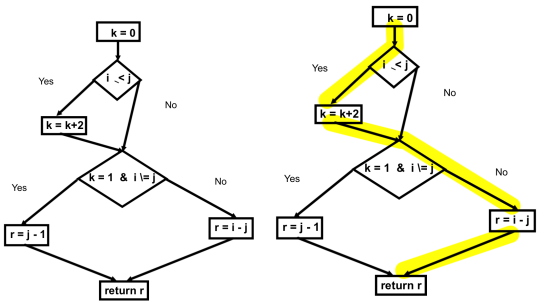


FIGURE 2 – Le CFG et chemin erroné – Le programme `AbsMinus`

$1 \wedge i \neq j$).

Puis, LocFaults tente de dévier une seconde condition. Le seul chemin possible est celui où les deux conditions du programme `AbsMinus` sont déviées. Cependant, comme il a le même préfixe que le premier chemin dévié, nous le rejetons.

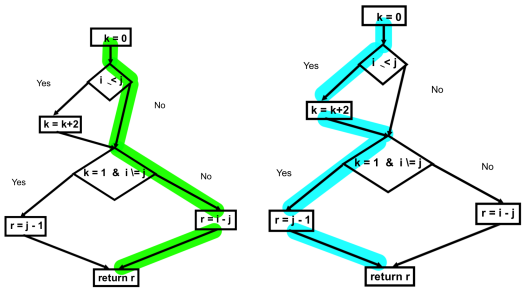


FIGURE 3 – Les chemins avec une déviation – Le programme `AbsMinus`

Cet exemple montre que LocFaults produit des informations pertinentes et utiles sur chaque chemin erroné. Contrairement à BugAssist [3, 4], un système de l'état de l'art, il ne fusionne pas toutes les instructions suspectes dans un seul ensemble, ce qui peut être difficile à exploiter par l'utilisateur.

Les entrées de notre algorithme sont le CFG du programme, CE : un contre-exemple, b_{cond} : une borne sur le nombre de conditions qui sont déviées, et b_{mcs} : une borne sur le nombre de MCSs(Minimal Correction Subsets) générés. Grosso modo, notre algorithme explore en profondeur le CFG en utilisant CE pour prendre la branche *If* ou *Else* de chaque nœud conditionnel, et collecte les contraintes qui correspondent aux affectations sur le chemin induit. Il dévie zéro, une ou au plus b_{cond} décisions par rapport au comportement du contre-exemple CE. À la fin d'un chemin, l'ensemble des contraintes qui ont été collectées est inconsistant, et au plus b_{mcs} MCSs sont calculés sur ce CSP.

Plus précisément, LocFaults procède comme suit :

* Il propage premièrement CE sur le CFG jusqu'à la fin du chemin initial erroné. Puis, il calcule au plus b_{mcs} MCSs sur le CSP courant. Ce qui représente une première localisation sur le chemin du contre-exemple.
* Après, LocFaults essaye de dévier une condition. Lorsque le premier nœud conditionnel (noté $cond$) est atteint, LocFaults prend la décision opposée à celle induite par CE, et continue à propager CE jusqu'au dernier nœud dans le CFG. Si le CSP construit à partir du chemin dévié est consistant, il y a deux types d'ensemble d'instructions suspectes :
  - le premier est la condition $cond$ elle-même. En effet, changer la décision pour $cond$ permet à CE de satisfaire la postcondition ;
  - une autre cause possible de l'erreur est une ou plusieurs mauvaises affectations avant $cond$ qui ont produit une décision erronée. Puis, LocFaults calcule aussi au plus b_{mcs} MCSs sur le CSP qui contient les contraintes collectées sur le chemin qui arrivent à $cond$.

  Ce processus est répété sur chaque nœud conditionnel du chemin du contre-exemple.
* Un processus similaire est ensuite appliqué pour dévier pour tout $k \leq b_{cond}$ conditions. Pour améliorer l'efficacité, les nœuds conditionnels qui corrigent le programme sont marqués avec le nombre de déviations qui ont été faites avant d'avoir été atteints. Pour une étape donnée k, si le changement de la décision d'un nœud conditionnel $cond$ marqué avec la valeur k' avec $k' \leq k$ corrige le programme, cette condition est ignorée. En d'autres termes, nous considérons seulement la première fois où un nœud conditionnel corrige le programme.

Cet algorithme incrémental basé sur les flots est un bon moyen pour aider le programmeur à la chasse aux bugs car il localise les erreurs autour du chemin du contre-exemple. Nos résultats ont confirmé que les temps de cet algorithme sont meilleurs par rapport à ceux qui correspondent à l'algorithme que nous avons présenté aux JFPC 2014 dans [2] [2]. Les sous-ensembles d'instructions suspectes fournis sont plus pertinents pour l'utilisateur. Dans le cadre de notre travaux futurs, nous envisageons de confirmer nos résultats sur des programmes avec plusieurs boucles complexes (voir nos premiers résultats dans [6]). Nous envisageons de comparer les performances de LocFaults avec des méthodes statistiques existantes ; par exemple : Tarantula [7, 8], Ochiai [9], AMPLE [9], Pinpoint [10]. Nous développons une version interactive de notre outil qui fournit les sous-ensembles suspects l'un après l'autre : nous voulons tirer profit des connaissances de l'utilisateur pour sélectionner les conditions qui doivent être déviées. Nous réfléchissons sur comment étendre notre méthode pour supporter les instructions numériques avec calcul sur les flottants.

2. Les résultats qui présentent les temps de calcul des deux versions de LocFaults, non-incrémentale et incrémentale, pour des programmes sans boucles sont disponibles à l'adresse `http://www.i3s.unice.fr/~bekkouch/Bench_Mohammed.html#rsba`

Références

[1] Bekkouche, Mohammed, Hélène Collavizza, and Michel Rueher. "LocFaults : A new flow-driven and constraint-based error localization approach*." SAC'15, SVT track.

[2] Bekkouche, Mohammed, Hélène Collavizza, and Michel Rueher. "Une approche CSP pour l'aide à la localisation d'erreurs." Dixièmes Journées Francophones de Programmation par Contraintes (JFPC 14).

[3] Jose, Manu, and Rupak Majumdar. "Cause clue clauses : error localization using maximum satisfiability." ACM SIGPLAN Notices 46.6 (2011) : 437-446.

[4] Jose, Manu, and Rupak Majumdar. "Bug-Assist : assisting fault localization in ANSI-C programs." Computer Aided Verification. Springer Berlin Heidelberg, 2011.

[5] Barnett, Mike, and K. Rustan M. Leino. "Weakest-precondition of unstructured programs." ACM SIGSOFT Software Engineering Notes. Vol. 31. No. 1. ACM, 2005.

[6] Bekkouche, Mohammed. "Exploration of the scalability of LocFaults approach for error localization with While-loops programs." arXiv preprint arXiv :1503.05508. 2015

[7] Jones, James A., and Mary Jean Harrold. "Empirical evaluation of the tarantula automatic fault-localization technique." Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM, 2005.

[8] Jones, James A., Mary Jean Harrold, and John Stasko. "Visualization of test information to assist fault localization." Proceedings of the 24th international conference on Software engineering. ACM, 2002.

[9] Abreu, Rui, Peter Zoeteweij, and Arjan JC Van Gemund. "On the accuracy of spectrum-based fault localization." Testing : Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007. IEEE, 2007.

[10] Chen, Mike Y., et al. "Pinpoint : Problem determination in large, dynamic internet services." Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on. IEEE, 2002.

[11] Liffiton, Mark H., and Karem A. Sakallah. "Algorithms for computing minimal unsatisfiable subsets of constraints." Journal of Automated Reasoning 40.1 (2008) : 1-33.

[12] Liffiton, Mark H., and Ammar Malik. "Enumerating infeasibility : Finding multiple muses quickly." Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer Berlin Heidelberg, 2013. 160-175.

Programmation par contraintes pour la vendange sélective

Nicolas Briot[1] **Christian Bessiere**[1] **Philippe Vismara**[1,2]

[1] LIRMM, Université de Montpellier, 161 rue Ada, 34095 Montpellier, France
[2] MISTEA, Montpellier SupAgro, INRA, 2 place Viala, 34060 Montpellier, France
{briot,bessiere,vismara}@lirmm.fr

Résumé

En viticulture de précision, la vendange sélective consiste à récolter séparément deux qualités de raisin dans une même vigne. On dispose pour cela d'une machine à vendanger comportant deux trémies et on connait la localisation des zones de qualités ainsi qu'une estimation des quantités à récolter. Le problème consiste à optimiser le trajet de la machine à vendanger tout en respectant de nombreuses contraintes comme la prise en compte du sens de traitement des rangs, la capacité de stockage de la machine, son rayon de braquage, le nombre de vidanges, etc. Après une formalisation du problème de la vendange sélective, cet article présente une modélisation sous la forme d'un problème de satisfaction de contraintes. La dernière section donne des résultats préliminaires fournis par le solveur AbsCon.

Abstract

In precision viticulture, the problem of differential harvesting occurs on vineyard with two qualities of grapes. Given estimated areas of different quality and quantity, the objective is to optimize the routing of the grape harvester under several constraints. This constraints are : harvest a certain amount of first quality grapes, capacity of the machine, turning radius, number of emptying, etc. After describing the problem of differential harvest, the paper models it as a constraint satisfaction problem. The last section is devoted to preliminary results with AbsCon solver.

1 Introduction

En viticulture de précision, de nombreuses études ont proposé de définir des zones de qualité intra-parcellaire [11]. Ces zones peuvent être déterminées à partir de photos aériennes, de capteurs en champ, de prélèvements ou des récoltes antérieures. Prendre en compte ces zones de qualité intra-parcellaire permet d'optimiser le coût et la qualité des vendanges. Cette approche a justifié le récent développement de prototypes de machine à vendanger capables de gérer deux qualités différentes de grappes, comme le prototype EnoControlTM system (NewHolland Agriculture, PA, USA). Ce type de machine possède deux réservoirs appelés *trémies* qui permet de stocker deux qualités différentes lors d'une même récolte.

Optimiser la récolte consiste à minimiser le temps de travail de la vendangeuse, ce qui correspond aux temps de trajets et aux temps de vidanges de la machine. Idéalement, le but à atteindre est de remplir les deux trémies de manière équilibrée afin de vidanger le moins possible la machine à vendanger. En effet, suivant la répartition des zones de qualité dans le vignoble et le rendement des vignes, la trémie qui contiendra une certaine qualité peut se remplir plus vite que l'autre. En plus des contraintes de capacité, il faut prendre en compte le rayon de braquage de la machine : plus le saut de rangs est petit, plus la machine doit manoeuvrer et perd du temps. Toutes ces contraintes rendent le problème combinatoire et complexe à résoudre. Des problèmes semblables, comme le problème du voyageur de commerce [1, 7] ou les problèmes de tournées de véhicules ont déja pu être traités. En programmation par contraintes, une étude de problèmes de tournées de véhicules a ainsi été étudiée en [8]. D'autres approches moins génériques, comme la recherche opérationnelle, ont été testées notamment en agriculture [2, 3]. Toutefois, le problème de la vendange sélective ne rentre dans aucun des problèmes déjà étudiés et reste inédit. Devant la multitude des contraintes et les différentes variantes possibles du modèle, l'utilisation de la programmation par contraintes semble pertinente afin de modéliser et de résoudre le problème.

Après une formalisation du problème de la vendange sélective, ce papier présente une modélisation de celui-ci sous la forme d'un problème d'optimisation de contraintes. Enfin, des résultats préliminaires fournis par le solveur AbsCon sont présentés en dernière partie.

2 Le problème de la vendange sélective

Le problème de la vendange sélective (ou *Differential Harvest Problem*) se pose sur un vignoble composé de plusieurs zones classées en deux catégories : les *A-zones* qui contiennent les *A-grappes*, c'est-à-dire les raisins de qualité supérieure et les *B-zones*, qui produisent le reste.

On considère une machine à vendanger (ou vendangeuse) qui possède deux réservoirs identiques appelés *trémies*, qui ont chacune une capacité maximale notée *Cmax*. Grâce à ce type de machine, les deux catégories de grappes peuvent être séparées et donc la qualité supérieure préservée. Dès le début de la récolte, une des trémies est dédiée à recevoir seulement les A-grappes (trémie a) et l'autre le reste. Une fois que la machine a rempli une de ses trémies, elle doit vidanger ses deux réservoirs dans des bennes situées autour de la parcelle. Généralement, les bennes sont emmenées en cave immédiatement après chaque vidange afin de préserver la qualité des grappes. Elles sont immédiatement remplacées par d'autres bennes en attentes sur la parcelle.

De par sa conception, la vendangeuse met un certain temps ($\approx$ 10 sec) pour acheminer le fruit pris de la vigne jusqu'à une de ses trémies. Ce temps est appelé *latence*. Ainsi, au moment où la machine à vendanger passe d'une zone à l'autre, la qualité du raisin récolté ne peut pas être garantie, car la latence impose un mixage des raisins récoltés à ce moment-là. Deux scénarios sont alors possibles : soit la machine passe d'une A-zone à une B-zone, alors les raisins ramassés dès l'entrée de la nouvelle zone sont considérés comme B-grappes et vont dans la trémie b ; soit la machine passe d'une B-zone à une A-zone et dans ce cas, les grappes (situées sur la A-zone) sont considérées comme B-grappes durant le temps de latence et iront dans la trémie correspondante. Ainsi pour un même rang, le type et le nombre de transitions peuvent être différents suivant le sens de récolte. La quantité de raisin de chaque catégorie peut dont varier. Par exemple, soit un rang ayant pour séquence les zones : $A-B-A-B$ (voir figure 1) : si la machine récolte le rang de gauche à droite, il y a seulement une transition $B-A$; inversement, si la machine passe de droite à gauche, il y a cette fois deux transitions $B-A$ et donc deux fois plus de A-grappes dans la trémie b.

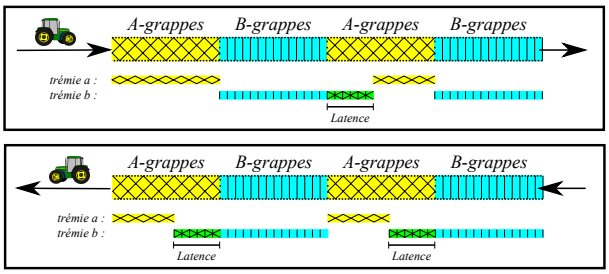


FIGURE 1 – Les quantités de grappes dépendent du sens de récolte du rang.

Notons *Rmin*, le volume de raisin de qualité A à atteindre. *Rmin* correspond à un volume minimal désiré par la cave pour compléter un manque de raisin de première catégorie. La vendange se déroule en deux phases. La première phase impose une vendange sélective jusqu'à ce que le seuil *Rmin* soit atteint. Pendant cette phase, la trémie a doit contenir uniquement du raisin A. La deuxième phase débute après la dernière vidange des trémies qui a permis d'atteindre le volume *Rmin* de raisin de qualité A. À partir de là, les deux trémies peuvent contenir les deux catégories de raisin sans aucune distinction. Suivant *Rmin*, il y a trois possibilités pour remplir les deux trémies : dans un premier temps, les deux trémies servent à séparer les deux catégories de raisins (voir figure 2.a). Dans le cas où la trémie a serait pleine, il est possible de continuer la récolte en dirigeant les raisins A dans l'autre trémie (voir figure 2.b). Ces raisins doivent être considérés par la suite comme du raisin déclassé. Enfin, lorsque *Rmin* est atteint, les deux trémies servent à récolter les grappes sans distinction (voir figure 2.c).

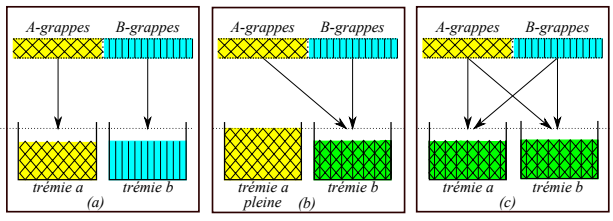


FIGURE 2 – 3 cas possibles pour remplir les trémies. (a) A-grappes et B-grappes sont séparées. (b) quand la trémie a est pleine, les A-grappes et B-grappes sont mixées dans la trémie b. (c) Une fois que *Rmin* est atteint, A-grappes et B-grappes sont mélangées dans les deux trémies.

La vigne, composée de n rangs, est modélisée en considérant les deux extrémités de chaque rang. Un rang

$i \in \{0, \ldots, n-1\}$ est représenté par ses extrémités $2i$ et $2i+1$. Pour chaque rang, notons $Q^A_{2i \to 2i+1}$ et $Q^B_{2i \to 2i+1}$ (resp. $Q^A_{2i+1 \to 2i}$ et $Q^B_{2i+1 \to 2i}$) les quantités de grappes contenues dans le rang i lorsque la machine récolte dans le sens $2i \to 2i+1$ (resp. $2i+1 \to 2i$). Ces quantités tiennent compte de la latence évoquée plus haut. Une autre donnée importante est le temps de trajet de la machine entre chaque rang et la benne. Notons $d(p,q) = d(q,p)\ \forall p, q \in \{0, 1, \ldots, 2n-1\} \cup \{2n\}$ le temps de passage d'une extrémité p à une extrémité q (avec l'extrémité $2n$ qui représente l'emplacement de la benne). Ce coût dépend du rayon de braquage de la machine à vendanger.

Définition 1. *Problème de la Vendange Sélective : Soit un vignoble représenté par une matrice de coût entre extrémités de rangs (ou benne), une estimation de la quantité de raisin A et B dans chaque rang suivant l'orientation, un seuil Rmin de A-grappes à atteindre et une machine à vendanger à deux trémies de capacité maximale $2 \times Cmax$. Le problème de la vendange sélective consiste à trouver une séquence de rangs qui minimise le déplacement total de la vendangeuse et qui permet de récolter tout le raisin dont au moins Rmin A-grappes.*

3 Modélisation CSP

Nous pouvons modéliser le problème de la vendange sélective comme un problème de satisfaction de contraintes ou *COP* décrit par : un ensemble de variables, un domaine représentant les valeurs possibles de chaque variable et un ensemble de contraintes sur ces variables. Le problème d'optimisation de contraintes consiste à rechercher une instanciation des variables avec une valeur de leurs domaines qui vérifie toutes les contraintes du problème tout en optimisant une fonction objectif.

Soit n le nombre de rangs du vignoble, λ le nombre passages à la benne avec γ le nombre de vidanges où le raisin est trié ($\gamma \leq \lambda \leq n$). Nous appelons *site* un rang ou une benne. Notons $R = \{0, \ldots, n-1\}$, l'ensemble des rangs et $\mathcal{S} = R \cup \{n, \ldots, n+\lambda-1\}$ l'ensemble des sites du vignoble. Le problème de la vendange sélective peut-être défini comme suit :

3.1 Variables

Nous créons, pour chaque site $i \in \mathcal{S}$:

- P_i et S_i les variables qui représentent respectivement le site précédant, et le site suivant le site i. Le domaine est $D(P_i) = D(S_i) = \mathcal{S} \setminus \{i\}$;
- Mix_i, la variable booléenne qui représente le mode de récolte. Si $Mix_i = 0$, les A-grappes sont préservées (première phase) sinon, ces grappes sont mélangées avec les B-grappes (deuxième phase). Le domaine de cette variable est : $D(Mix_i) = \{0, 1\}$;
- U^A_i et U^B_i, représentent les quantités totales de A-grappes et de B-grappes récoltées depuis la dernière vidange jusqu'au site i inclus. On a : $D(U^A_i) = D(U^B_i) = \{0, \ldots, 2 \times Cmax\}$. La variable U^B_i peut dépasser $Cmax$ uniquement en mode non-trié ;
- T_i, la variable représentant le temps de trajet entre le site i et le site précédent. $D(T_i) = \mathbb{N}$;

Nous créons, pour chaque rang $r \in R$:

- Ori_r, la variable booléenne représentant l'orientation du rang r (0 représente la direction $2r \to 2r+1$ et 1 l'inverse). $D(Ori_r) = \{0, 1\}$;
- u^A_r (resp. u^B_r), la variable représentant la quantité de raisin de qualité A (resp. B) contenue dans le rang r suivant son orientation. $D(u^A_r) = D(u^B_r) = \mathbb{N}$;

3.2 Contraintes

La première contrainte est la contrainte globale alldifferent (1) qui contraint de passer une et une seule fois dans chaque rang. Elle porte sur l'ensemble des variables précédant :

$$AllDifferent(P_0, \ldots, P_{n+\lambda-1}) \tag{1}$$

La contrainte (2) est une contrainte de channeling entre les variables précédant et suivant :

$$P_{S_i} = i \qquad S_{P_i} = i \qquad \forall i \in \mathcal{S} \tag{2}$$

Les contraintes (3) et (4) imposent le mode de vendange (sélective ou pas) suivant l'indice correspondant à une benne de la phase 1 (n à $n+\gamma-1$) ou de la phase 2 ($n+\gamma$ à $n+\lambda-1$). La contrainte (5) est une containte « element » qui transmet le mode de vendange entre successeurs.

$$Mix_i = 0 \qquad \forall i \in \{n, \ldots, n+\gamma-1\} \tag{3}$$

$$Mix_i = 1 \qquad \forall i \in \{n+\gamma, \ldots, n+\lambda-1\} \tag{4}$$

$$Mix_r = Mix_{S_r} \qquad \forall r \in R \tag{5}$$

La contrainte de table suivante permet d'attribuer les quantités de raisin suivant le sens de traitement du rang. $\forall i \in R$, on a :

Ori_i	u^A_i	u^B_i
0	$Q^A_{2i \to 2i+1}$	$Q^B_{2i \to 2i+1}$
1	$Q^A_{2i+1 \to 2i}$	$Q^B_{2i+1 \to 2i}$

Les quantités de raisin récoltées (contenues dans les trémies) depuis une vidange sont calculées en fonction

du volume contenu dans le rang et de la quantité précédente. Les quantités associées à une benne valent 0 :

$$U_i^\alpha = 0 \qquad \forall \alpha \in \{A, B\} \;\; \forall i \in \mathcal{S} \setminus R \qquad (6)$$
$$U_i^\alpha = U_{P_i}^\alpha + u_i^\alpha \qquad \forall \alpha \in \{A, B\} \; \forall i \in R \qquad (7)$$

Les quantités calculées en (7) sont limitées par la capacité des trémies (8). La quantité maximale de B-grappes diffère suivant le mode de récolte (9).

$$U_i^A + U_i^B \leq 2 \times Cmax \qquad \forall i \in R \qquad (8)$$
$$U_i^B \leq (1 + Mix_i) \times Cmax \qquad \forall i \in R \qquad (9)$$

La contrainte (10) permet de remplir le contrat de récolte $Rmin$. Elle concerne uniquement le mode de récolte séparé, c'est-à-dire les vidanges de n à $n + \gamma - 1$. Il faut prendre en compte que les A-grappes qui sont dans la trémie a. Cette quantité correspond à la partie de U_i^A qui ne dépasse pas la capacité de la trémie a.

$$\sum_{i=n}^{n+\gamma-1} min(U_{p_i}^A, Cmax) \geq Rmin \qquad (10)$$

Il est possible de reformuler la contrainte (10) sans la fonction *minimum*. Nous ajoutons une nouvelle variable nommée $Ac_i \; \forall i \in \{n, \ldots, n+\gamma-1\}$ de domaine $D(Ac_i) = \{0, \ldots, Cmax\}$ avec :

$$Ac_i \leq Cmax \qquad (10'a)$$
$$Ac_i \leq U_{P_i}^A \qquad \forall i \in \{n, \ldots, n+\gamma-1\} \qquad (10'b)$$
$$\sum_{i=n}^{n+\gamma-1} Ac_i \geq Rmin \qquad (10'c)$$

Il est possible d'interdire les trajets entre deux extrémités qui ne sont pas du même côté du vignoble par la contrainte (11) :

$$Ori_i = 1 - Ori_{P_i} \qquad \forall i \in R \qquad (11)$$

Enfin, l'optimisation porte sur le temps de trajet de la machine. On pose la contrainte (12), que l'on codera par une contrainte de table entre le site précédent, son orientation et le site i :

$$T_i = d(2P_i + Ori_{P_i}, 2i + (1 - Ori_i)) \quad \forall i \in \mathcal{S} \qquad (12)$$

Au final, on souhaite optimiser l'expression suivante :

$$Minimiser \; : \; \sum_{i=0}^{n+\lambda} T_i \qquad (13)$$

4 Améliorations du modèle

Nous avons présenté dans la section précédente une modélisation du problème de la vendange sélective. Nous avons opté pour des variables représentant les rangs précédents et suivants. Cette modélisation possède une symétrie de valeurs. Elle est liée aux bennes de même type qui sont interchangeables. Ces permutations donnent des solutions de même coût. Afin d'éliminer ces symétries, nous ajoutons au modèle les contraintes suivantes :

$$P_i < P_{i+1} \qquad \forall i \in \{n, \ldots, n+\gamma-2\} \qquad (14)$$
$$P_i < P_{i+1} \qquad \forall i \in \{n+\gamma, \ldots, n+\lambda-1\} \qquad (15)$$

La contrainte (14) (respectivement (15)) impose un ordre croissant sur les valeurs des rangs précédant une vidange séparée (resp. non-séparée).

Par ailleurs, comme il est dit dans [8] pour les problèmes de tournées de véhicules, le modèle proposé dans la section 3 présente un autre défaut. Lors de la recherche, des cycles ne passant par aucune benne risquent d'être explorés inutilement. Il est possible d'interdire cette configuration par la contrainte globale *cycle*. Cette contrainte, introduite dans [5] et [10] permet, en ayant un ensemble de variables représentant des successeurs (ou prédécesseurs) d'avoir un certain nombre de cycles entre elles. Ici, nous imposons qu'il y ait un seul cycle sur les variables représentant les sites précédents et un seul sur les variables représentant les sites suivants. La combinaison de la contrainte *cycle* avec la contrainte $Alldifferent$ permet d'avoir un seul et unique prédécesseur et successeur. Dans notre cas, une modélisation avec ou sans cette contrainte possède le même ensemble de solution mais l'efficacité du filtrage est alors amoindrie.

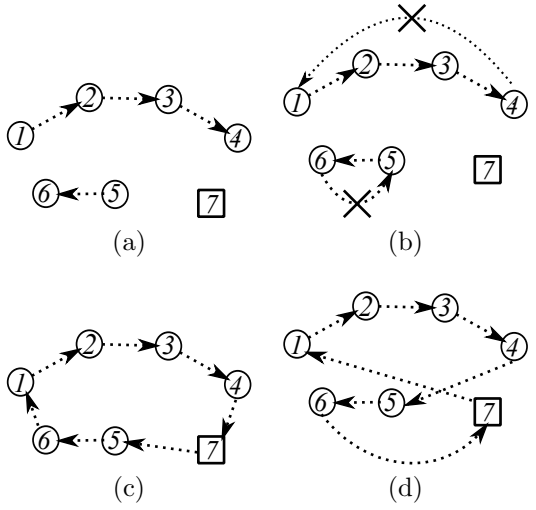


FIGURE 3 – La contrainte de cycle sur les variables successeurs entre les rangs (cercles) et une benne (carré).

Par exemple, dans la figure 3, à partir de la solution partielle (a), les cycles présents dans la solution (b)

peuvent être filtrés. Autrement dit, imposer un seul cycle permet de retirer les valeurs (figure 3.b) : $1 \in \mathrm{D}(S_4)$ et $4 \in \mathrm{D}(P_1)$, $5 \in \mathrm{D}(S_6)$ et $6 \in \mathrm{D}(P_5)$ ($7 \notin \mathrm{D}(S_7)$ et $\mathrm{D}(P_7)$) et les seules solutions sont données en (c) et (d). A notre modélisation, nous ajoutons les contraintes (16) et (17) :

$$Cycle(1, (P_0, \ldots, P_{n+\lambda})) \quad (16)$$
$$Cycle(1, (S_0, \ldots, S_{n+\lambda})) \quad (17)$$

5 Travaux connexes

Une première modélisation de ce problème a été présentée dans [4] avec laquelle nous avons pu comparer les résultats. Dans cette précédente modélisation, une variable est associée à chaque étape pour décrire l'extrémité visitée à cette étape. Des variables booléennes, associées à chaque étape, indiquent si la machine doit faire une vidange à ce moment-là ou non. Le basculement entre la récolte sélective et la récolte normale est imposé en fixant l'indice du rang après lequel la récolte devient simple. Ainsi, la recherche consiste à lancer plusieurs résolutions en parallèles avec les différents indices possibles pour le basculement et à conserver le résultat optimal. Dans cet article, nous avons montré l'interêt de la démarche qui permet jusqu'à 40% de gain de temps par rapport à une trajectoire traditionelle de la machine à vendanger.

6 Résultats

Nous avons implémenté les deux modélisations avec le solveur AbsCon [9], écrit en Java, auquel nous avons adjoint une contrainte *cycle* garantissant une permutation circulaire entre les variables. Nos tests ont été effectués sur un MacBook Pro avec 2,6 GHz Intel Core i5 et 8 Go de RAM. Les données ont été fournies par Montpellier SupAgro et sont issues d'un vignoble provenant de l'INRA du Pech-Rouge (Gruissan, Aude, France) (figure 4). La table 1 représente le temps de calcul et le nombre de nœuds pour la résolution du problème avec les deux modèles. Le premier modèle correspond à celui présenté dans [4]. Les colonnes 2 et 3 correspondent à la modélisation présentée ici sans et avec les contraintes de cycle. Les nombres de rangs testés (n=10 et n=16) correspondent à un sous-ensemble de rangs de la même parcelle. Pour chaque modèle, nous avons utilisé l'heuristique de choix de valeurs *Random*, qui donne les meilleurs temps de calculs et nous avons imposé un cutoff de 10 échecs au départ, multiplié par 2 à chaque restart.

Ces résultats montrent que sur de petites instances réelles, le modèle proposé permet de trouver la solution optimale dans un temps relativement raisonnable

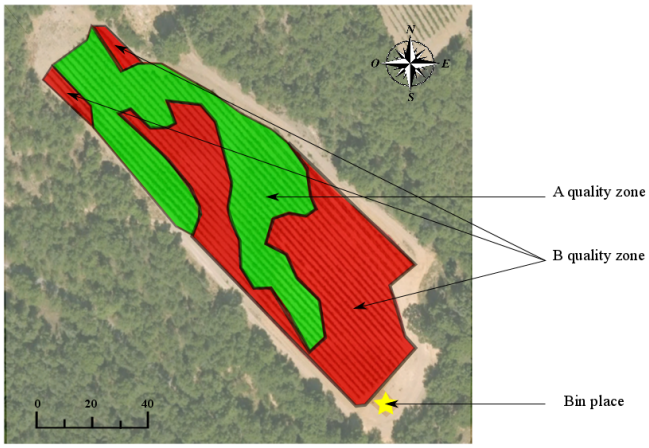


FIGURE 4 – Un vignoble avec deux qualités : B-grappes en rouge et A-grappes en vert.

et améliore sensiblement les temps de calculs du modèle présenté dans [4]. L'ajout des contraintes de cycle permet de diminuer l'espace de recherche et le temps de résolution par rapport à la même approche sans ces contraintes. Ces résultats restent préliminaires et doivent être confrontés au passage à l'échelle, mais sont encourageants pour la suite. De plus, il est important de noter que notre approche est exacte et calcule la solution optimale du problème. Pour de plus grandes instances, nous pourrions recourir à des méthodes locales de type LNS (Large Neigborhood Search) qui ont donné de bons résultats sur un problème similaire [6].

7 Conclusion

Dans cet article, nous avons présenté le problème de la vendange sélective issu de l'agriculture de précision. Après une formalisation de ce problème, une modélisation sous la forme d'un problème d'optimisation sous contraintes à été présentée. Ce nouveau modèle permet de résoudre le problème de manière optimale en des temps raisonnables sur de petites instances réelles. Il reste à prouver la validité de l'approche sur de plus grandes instances.

	Modèle de [4]		Modèle sans (16) et (17)		Modèle avec (16) et (17)	
n	tps	# Noeuds	tps	# Noeuds	tps	# Noeuds
10	1 100s	6 624 876	413s	48 050	41s	8 038
16	326 143s	647 119 328	67 826s	8 688 417	11 841s	177 894

TABLE 1 – Résultats sur 10 et 16 rangs. Comparaison entre le modèle de [4] et celui proposé dans cet article.

Références

[1] Pascal Benchimol, Willem Jan van Hoeve, Jean-Charles Régin, Louis-Martin Rousseau, and Michel Rueher. Improved filtering for weighted circuit constraints. *Constraints*, 17(3) :205–233, 2012.

[2] Dionysis Bochtis and Claus G. Sørensen. The vehicle routing problem in field logistics : part i. *Biosystems engineering*, 104 :447–457, 2009.

[3] Dionysis Bochtis and Claus G. Sørensen. The vehicle routing problem in field logistics : part ii. *Biosystems engineering*, 105 :180–188, 2010.

[4] Nicolas Briot, Christian Bessiere, Bruno Tisseyre, and Philippe Vismara. Integration of operational constraints to optimize differential harvest in viticulture. In *Proc. 10th European Conference on Precision Agriculture*, July 2015, à paraitre.

[5] Yves Caseau and François Laburthe. Solving small tsps with constraints. In Lee Naish, editor, *Logic Programming, Proceedings of the Fourteenth International Conference on Logic Programming, Leuven, Belgium, July 8-11, 1997*, pages 316–330. MIT Press, 1997.

[6] Luca Di Gaspero, Andrea Rendl, and Tommaso Urli. Constraint-based approaches for balancing bike sharing systems. In *Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 758–773. Springer, 2013.

[7] Jean-Guillaume Fages. *Exploitation de structures de graphe en programmation par contraintes.* PhD thesis, Ecole des Mines de Nantes, 2014.

[8] Philip Kilby and Paul Shaw. Vehicle routing. In Francesca Rossi, Peter Van Beek, and Toby Walsh, editors, *Handbook of constraint programming*, chapter 23, pages 799, 834. Elsevier, 2006.

[9] Sylvain Merchez, Christophe Lecoutre, and Frédéric Boussemart. Abscon : A prototype to solve csps with abstraction. In Toby Walsh, editor, *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, Paphos, Cyprus, 2001*, volume 2239 of *Lecture Notes in Computer Science*, pages 730–744. Springer, 2001.

[10] Gilles Pesant, Michel Gendreau, Jean-Yves Potvin, and Jean-Marc Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32(1) :12–29, 1998.

[11] Bruno Tisseyre, Hernan Ojeda, and James Taylor. New technologies and methodologies for site-specific viticulture. *Journal International des Sciences de la Vigne et du Vin*, 41 :63–76, 2007.

Autour des Triangles Cassés *

Martin C. Cooper[1] **Achref El Mouelhi**[2] **Cyril Terrioux**[2] **Bruno Zanuttini**[3]

[1] IRIT, Université de Toulouse III, 31062 Toulouse, France
[2] LSIS, Aix-Marseille Université, 13397 Marseille, France
[3] GREYC, Normandie Université, 14032 Caen, France
cooper@irit.fr {achref.elmouelhi, cyril.terrioux}@lsis.org
bruno.zanuttini@unicaen.fr

Résumé

Une instance CSP binaire qui satisfait la propriété des triangles cassés (BTP) peut être résolue en temps polynomial. Malheureusement, en pratique, peu d'instances satisfont cette propriété. Nous montrons qu'une version locale de BTP permet de fusionner des valeurs dans les domaines d'instances binaires quelconques. Des expérimentations démontrent la diminution significative de la taille de l'instance pour certaines classes de problèmes. Ensuite, nous proposons une généralisation de cette fusion à des contraintes d'arité quelconque. Enfin, une version orientée nous permet d'étendre la classe polynomiale BTP.

Ce papier est un résumé de l'article M. C. Cooper, A. El Mouelhi, C. Terrioux et B. Zanuttini. On Broken Triangles. In *Proceedings of CP, LNCS* 8656, 9–24, 2014.

Abstract

A binary CSP instance satisfying the broken-triangle property (BTP) can be solved in polynomial time. Unfortunately, in practice, few instances satisfy the BTP. We show that a local version of the BTP allows the merging of domain values in arbitrary instances of binary CSP. Experimental trials demonstrate a significant decrease in instance size for certain classes of problems. Then we propose a generalization of this merging to instances with constraints of arbitrary arity. Finally, a directional version allows us to extend the BTP tractable class .

This is a summary of the paper M. C. Cooper, A. El Mouelhi, C. Terrioux et B. Zanuttini. On Broken Triangles. In *Proceedings of CP, LNCS* 8656, 9–24, 2014.

1 Fusion de valeurs pour les CSP binaires

Nous considérons ici une nouvelle opération de réduction de domaines, qui au lieu d'éliminer des valeurs comme le ferait une opération classique comme la cohérence d'arc ou la substitution de voisinage, fusionne des valeurs issues d'un même domaine d'un CSP binaire. Nous rappelons qu'un *CSP binaire* est défini par un ensemble X de n variables, un domaine $\mathcal{D}(x)$ de valeurs possibles pour chaque variable $x \in X$ et une relation $R_{xy} \subseteq \mathcal{D}(x) \times \mathcal{D}(y)$ pour chaque couple de variables distinctes $x, y \in X$, qui représente l'ensemble des couples (a, b) de valeurs compatibles pour (x, y).

La *fusion* de deux valeurs $a, b \in \mathcal{D}(x)$ d'un CSP binaire consiste à remplacer a et b dans $\mathcal{D}(x)$ par une nouvelle valeur c qui est compatible avec toutes les valeurs des autres variables compatibles avec a ou b. Une *condition de fusion de valeurs* pour les valeurs $a, b \in \mathcal{D}(x)$ d'une instance I est une propriété $P(x, a, b)$ calculable en temps polynomial telle que, si la propriété est vraie, alors l'instance I' obtenue à partir de I en fusionnant $a, b \in \mathcal{D}(x)$ est satisfiable ssi I est satisfiable.

Nous définissons une condition de fusion de valeurs basée sur la propriété BTP [1]. Un *triangle cassé* sur deux valeurs $a, b \in \mathcal{D}(x)$ se définit par un couple de valeurs $d \in \mathcal{D}(y)$, $e \in \mathcal{D}(z)$ de deux variables différentes $y, z \in X \setminus \{x\}$ tel que $(a, d) \notin R_{xy}$, $(b, d) \in R_{xy}$, $(a, e) \in R_{xz}$, $(b, e) \notin R_{xz}$ et $(d, e) \in R_{yz}$ (voir figure 1).

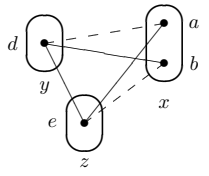


FIGURE 1 – Un triangle cassé sur deux valeurs a, b d'une variable x donnée.

Proposition 1 *L'absence de triangle cassé sur a et b est une condition de fusion de valeurs.*

*Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet TUPLES (ANR-2010-BLAN-0210).

La règle de fusion définie par la proposition 1 généralise la substitution de voisinage [3] et l'interchangeabilité virtuelle [4]. Par ailleurs, en plus de conserver la satisfiabilité, elle permet de reconstruire en temps polynomial toutes les solutions de I à partir des solutions d'une instance I^f obtenue en appliquant à I une suite de fusions. À noter que la reconstruction d'une solution de I à partir d'une solution de I^f peut s'effectuer en temps linéaire dans la taille de l'instance I.

Des expérimentations menées sur plus de 2 300 instance de la compétition [1] de solvers de 2008 ont montré que pour 10 % des instances, cette règle s'avère très efficace avec au moins 40 % de valeurs fusionnées.

2 Extension aux CSP d'arité quelconque

Nous généralisons maintenant la fusion de valeurs aux CSP d'arité quelconque. Une *instance CSP d'arité quelconque* I est définie par un ensemble X de n variables, un domaine $\mathcal{D}(x)$ de valeurs possibles pour chaque variable $x \in X$ et un ensemble NoGoods(I) de tuples incompatibles. La *fusion* de deux valeurs $a, b \in \mathcal{D}(x)$ d'un CSP d'arité quelconque consiste à remplacer a et b dans $\mathcal{D}(x)$ par une nouvelle valeur c qui est compatible avec tout tuple compatible avec a ou b. L'ensemble de nogoods de l'instance I' ainsi obtenue est donc défini par :

$$\begin{aligned}\text{NoGoods}(I') = \{&t \in \text{NoGoods}(I) \mid \langle x,a\rangle, \langle x,b\rangle \notin t\}\\ &\cup \{t \cup \{\langle x,c\rangle\} \mid t \cup \{\langle x,a\rangle\} \in \text{NoGoods}(I) \wedge\\ &\qquad \exists t' \in \text{NoGoods}(I) \text{ t.q. } t' \subseteq t \cup \{\langle x,b\rangle\}\}\\ &\cup \{t \cup \{\langle x,c\rangle\} \mid t \cup \{\langle x,b\rangle\} \in \text{NoGoods}(I) \wedge\\ &\qquad \exists t' \in \text{NoGoods}(I) \text{ t.q. } t' \subseteq t \cup \{\langle x,a\rangle\}\}.\end{aligned}$$

Nous pouvons alors définir une condition de fusion de valeurs basée sur une généralisation des triangles cassés. Un *triangle cassé d'arité générale* sur deux valeurs $a, b \in \mathcal{D}(x)$ est défini par un couple de tuples t, u ne contenant pas d'affectation de la variable x et satisfaisant les conditions suivantes :

1. $\text{Good}(I, t \cup u) \quad \wedge \quad \text{Good}(I, t \cup \{\langle x,a\rangle\}) \quad \wedge \quad \text{Good}(I, u \cup \{\langle x,b\rangle\})$
2. $t \cup \{\langle x,b\rangle\} \in \text{NoGoods}(I) \quad \wedge \quad u \cup \{\langle x,a\rangle\} \in \text{NoGoods}(I)$

où Good(I, t) est un prédicat qui est vrai ssi t ne contient pas deux affectations différentes d'une même variable et $\nexists t' \subseteq t$ tel que $t' \in \text{NoGoods}(I)$.

Proposition 2 *L'absence de triangle cassé d'arité générale sur a et b est une condition de fusion de valeurs.*

Comme dans le cas binaire, une solution de I peut être calculée en temps linéaire à partir d'une solution d'une instance I^f obtenue en appliquant à I une suite de fusions.

Notons qu'il existe un lien théorique entre la fusion de valeurs ne possèdant pas de triangle cassé d'arité quelconque et la résolution exploitée dans SAT.

1. http://www.cril.univ-artois.fr/CPAI08

3 Une nouvelle classe polynomiale

À l'image de la propriété BTP dans le cas des CSP binaires, nous pouvons considérer la notion de triangle cassé d'arité générale selon un ordre total $<$ sur les variables. Un *triangle cassé directionnel d'arité générale* sur deux valeurs $a, b \in \mathcal{D}(x)$ d'une instance I est un couple de tuples t, u ne contenant pas d'affectation de la variable x et satisfaisant les conditions suivantes :

1. $t^{<x}$ et $u^{<x}$ ne sont pas vides
2. $\text{Good}(I, t^{<x} \cup u^{<x}) \quad \wedge \quad \text{Good}(I, t^{<x} \cup \{\langle x,a\rangle\}) \quad \wedge \quad \text{Good}(I, u^{<x} \cup \{\langle x,b\rangle\})$
3. $\exists t'$ t.q. $Vars(t') = Vars(t) \quad \wedge \quad (t')^{<x} = t^{<x} \quad \wedge \quad t' \cup \{\langle x,a\rangle\} \notin \text{NoGoods}(I)$
4. $\exists u'$ t.q. $Vars(u') = Vars(u) \wedge (u')^{<x} = u^{<x} \wedge u' \cup \{\langle x,b\rangle\} \notin \text{NoGoods}(I)$
5. $t \cup \{\langle x,b\rangle\} \in \text{NoGoods}(I) \quad \wedge \quad u \cup \{\langle x,a\rangle\} \in \text{NoGoods}(I)$

avec $t^{<x}$ le tuple restreint aux variables précédant x dans l'ordre $<$ et $Vars(t)$ l'ensemble des variables sur lesquelles porte t. I satisfait la *propriété des triangles cassés directionnels d'arité quelconque* (DGABTP) selon l'ordre sur les variables $<$ s'il n'existe aucun triangle cassé directionnel sur a, b pour tout couple a, b de n'importe quelle variable x.

Cette propriété permet de définir une nouvelle classe polynomiale quand l'ordre sur les variables est connu.

Théorème 1 *Le problème de satisfiabilité est polynomial pour les instances CSP satisfaisant la propriété DGABTP selon un ordre fourni avec la donnée du problème.*

La reconnaissance de la classe DGABTP requiert donc de déterminer s'il existe un ordre sur les variables selon lequel l'instance CSP considérée satisferait la propriété DGABTP. Si dans le cas des CSP binaires, ce problème peut se résoudre en temps polynomial, il devient malheureusement NP-complet dans les cas des CSP d'arité quelconque.

Enfin, nous pouvons démontrer que les classes DGABTP et DBTP (pour Dual Broken Triangle Property [2]) sont incomparables.

Références

[1] M. C. Cooper, P. G. Jeavons, et A. Z. Salamon. Generalizing constraint satisfaction on trees : Hybrid tractability and variable elimination. *Artificial Intelligence*, 174(9-10), 570–584, 2010.

[2] A. El Mouelhi, P. Jégou et C. Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. In *ICTAI*, 947–954, 2013.

[3] E. C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *AAAI*, 227–233, 1991.

[4] C. Likitvivatanavong et R. H.C. Yap. Eliminating redundancy in CSPs through merging and subsumption of domain values. *ACM SIGAPP Applied Computing Review*, 13(2), 2013.

Une famille de classes polynomiales de CSP basée sur la microstructure *

Martin Cooper[1] **Philippe Jégou**[2] **Cyril Terrioux**[2]

[1] IRIT, University of Toulouse III,
31062 Toulouse, France
[2] Aix-Marseille Université, LSIS UMR CNRS 7296
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20, France
cooper@irit.fr {philippe.jegou, cyril.terrioux}@lsis.org

Résumé

L'étude des classes polynomiales constitue une question importante en intelligence artificielle, en particulier au niveau des problèmes de satisfaction de contraintes. Dans ce contexte, la propriété BTP fournit une classe importante de l'état de l'art. Dans cet article, nous proposons d'étendre et de généraliser cette classe en introduisant la propriété k-BTP (et la classe des instances satisfaisant cette propriété) où le paramètre k est une constante donnée. Ainsi, nous avons 2-BTP = BTP, et pour $k > 2$, k-BTP est une relaxation de BTP au sens où k-$BTP \subsetneq (k+1)$-BTP. En outre, nous montrons que si k-TW est la classe d'instances ayant une largeur arborescente bornée par une constante k, alors k-$TW \subsetneq (k{+}1)$-BTP. Au niveau de la complexité, nous montrons que les instances satisfaisant k-BTP et qui vérifient la k-cohérence-forte sont reconnaissables et résolubles en temp polynomial. Nous étudions aussi la relation entre k-BTP et l'approche de W. Naanaa qui a proposé un outil théorique connu sous le vocable *directional rank* afin d'étendre les classes polynomiales de manière paramétrée. Enfin, nous proposons une étude expérimentale de 3-BTP qui montre l'intérêt pratique de cette classe.

Abstract

The study of tractable classes is an important issue in Artificial Intelligence, especially in Constraint Satisfaction Problems. In this context, the Broken Triangle Property (BTP) is a state-of-the-art microstructure-based tractable class which generalizes well-known and previously-defined tractable classes. In this paper, we propose to extend and to generalize this class using a more general approach based on a parameter k which is a given constant. To this end, we introduce the k-BTP property (and the class of instances satisfying this property) such that we have 2-BTP = BTP, and for $k > 2$, k-BTP is a relaxation of BTP in the sense that k-$BTP \subsetneq (k+1)$-BTP. Moreover, we show that if k-TW is the class of instances having tree-width bounded by a constant k, then k-$TW \subsetneq (k{+}1)$-BTP. Concerning tractability, we show that instances satisfying k-BTP and which are strong k-consistent are tractable, that is, can be recognized and solved in polynomial time. We also study the relationship between k-BTP and the approach of Naanaa who proposed a set-theoretical tool, known as the directional rank, to extend tractable classes in a parameterized way. Finally we propose an experimental study of 3-BTP which shows the practical interest of this class.

1 Introduction

Identifier des fragments polynomiaux, généralement appelés *classes polynomiales*, est une question importante en intelligence artificielle, en particulier dans les problèmes de satisfaction de contraintes (CSP). De nombreuses études ont abordé cette question, notamment dès les débuts de l'intelligence artificielle. Ces résultats sont souvent de nature théorique avec, dans certains cas, la mise en évidence de classes polynomiales qui peuvent très souvent être considérées comme artificielles, au sens où il s'avère difficile, voire impossible, de les exploiter pour la résolution d'instances du monde réel. Cela étant, certaines classes polynomiales ont cependant effectivement été utilisées en

*Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet TUPLES (ANR-2010-BLAN-0210) ainsi que par l'EPSRC grant EP/L021226/1.

pratique, comme notamment les classes définies par les réseaux de contraintes de largeur arborescente bornée [6, 11]. Plus récemment, le concept de classe hybride a été introduit notamment avec la classe BTP [1]. Cette classe contient strictement des classes polynomiales structurelles (comme les CSP arborescents) et des classes polynomiales définies par restriction de langage. Un avantage majeur de cette classe, en plus de sa généralisation de classes polynomiales déjà connues, est lié à son intérêt pratique. En effet, les instances de cette classe peuvent être résolues en temps polynomial à l'aide d'algorithmes tels que MAC (Maintaining Arc-Consistency [17]) et RFL (Real Full Look-ahead [15]), algorithmes qui sont généralement mis en œuvre dans les solveurs. Cela permet à BTP d'être directement utilisée en pratique. En outre, cette particularité peut aussi aider à expliquer théoriquement l'efficacité pratique, souvent remarquable des solveurs, alors même que la complexité en temps des algorithmes qu'ils implémentent est exponentielle dans le pire des cas.

Dans cette contribution, nous revenons sur ce type d'approche en généralisant la classe polynomiale BTP dont la définition s'appuie sur l'exclusion de certains motifs (appelés *triangles cassés*) dans le graphe de microstructure associé à toute instance de CSP binaire. Des travaux très récents allant dans la même direction ont introduit la classe ETP [13] qui généralise BTP en relaxant certaines de ses conditions, puisque certains triangles cassés sont tolérés dans ETP alors qu'ils sont interdits pour BTP. Ici, nous proposons une généralisation plus large appelée k-BTP qui étend ces travaux selon deux axes. Tout d'abord, dans le même esprit que ETP, la nouvelle classe tolère la présence d'un nombre plus grand de triangles cassés, généralisant ainsi strictement ETP (et par conséquent BTP). Deuxièmement, la classe k-BTP est paramétrée par une constante k de sorte à offrir une version générique et donc plus large, qui montre son intérêt théorique pour des valeurs quelconques de k, bien qu'en pratique, le cas pour lequel $k = 3$ doit probablement constituer la classe la plus intéressante. Ainsi, alors que BTP est définie pour des ensembles de 3 variables et ETP pour des ensembles de 4 variables, k-BTP est définie sur la base d'ensembles de $k+1$ variables où k est une constante fixée. Dans cette approche, $BTP = 2\text{-}BTP$ tandis que $ETP \subsetneq 3\text{-}BTP$. Ainsi, cette approche rend possible une généralisation stricte de ces deux classes. De plus, k-BTP conserve certaines de leurs propriétés intéressantes ainsi que certains des avantages pratiques. Notamment, nous montrons que les algorithmes classiques comme MAC ou RFL peuvent résoudre des instances appartenant à k-BTP en temps polynomial, en supposant que ces instances vérifient la *k-cohérence-forte* [9]. En outre, nous mettons en évidence les relations de cette classe avec des classes polynomiales structurelles et hybrides de la littérature. Nous montrons en particulier que la classe des réseaux de contraintes dont la largeur arborescente est bornée par k est strictement incluse dans la classe $k+1$-BTP. Ce résultat donne une première réponse à une question posée très récemment par M. Vardi et qui portait sur les relations pouvant exister entre ETP et la classe polynomiale induite par les instances de largeur arborescente bornée [18]. Nous revenons également sur un résultat récent mais relativement méconnu qui a été proposé par W. Naanaa [14] et dont nous étudions les relations avec k-BTP.

Dans la partie 2, nous rappelons les définitions des classes polynomiales BTP et ETP. Dans la partie 3 nous définissons la nouvelle classe k-BTP et nous montrons que les instances de cette classe peuvent être reconnues en temps polynomial. En outre, nous montrons que sous l'hypothèse supplémentaire de vérification de la *k-cohérence-forte*, les instances vérifiant k-BTP peuvent être résolues en temps polynomial et que les algorithmes standards (comme MAC ou RFL) peuvent les résoudre polynomialement. Dans la partie 4 nous étudions les relations entre k-BTP et plusieurs classes polynomiales de la littérature, tandis que dans la partie 5, nous présentons des résultats expérimentaux sur la présence de cette classe au sein des benchmarks utilisés par la communauté, ainsi que sur la résolution de ses instances.

2 Préliminaires

Formellement, un *problème de satisfaction de contraintes* (*CSP*) aussi appelé *réseau de contraintes* est un triplet (X, D, C), où $X = \{x_1, \ldots, x_n\}$ est un ensemble de n variables, $D = (D_{x_1}, \ldots, D_{x_n})$ est une liste de domaines finis de valeurs, un par variable, et $C = \{c_1, \ldots, c_e\}$ est un ensemble de e contraintes. Chaque contrainte c_i est une paire $(S(c_i), R(c_i))$, où $S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ est la *portée* (ou *scope)* de c_i, et $R(c_i) \subseteq D_{x_{i_1}} \times \cdots \times D_{x_{i_k}}$ est sa *relation de compatibilité*. L'*arité* de c_i est $|S(c_i)|$. Dans cet article, nous ne considérerons que le cas des CSP binaires, c'est-à-dire des CSP pour lesquels toutes les contraintes ont pour arité 2. Pour simplifier la notation, nous noterons c_{ij} la contrainte portant sur x_i et x_j. La structure d'un réseau de contraintes est représentée par un graphe appelé *graphe de contraintes*, dont les sommets correspondent aux variables et les arêtes aux portées des contraintes. L'affectation des variables d'un sous-ensemble Y de X est dite *cohérente* si elle ne viole aucune contrainte dont la portée est incluse dans Y. Nous utiliserons la notation $R(c_{ij})[a]$ pour représenter l'ensemble des

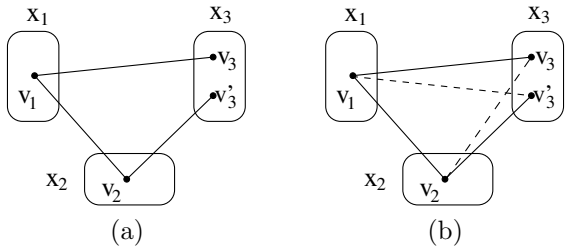


FIGURE 1 – Une instance ne vérifiant pas BTP (a) et une instance BTP (b) par rapport à l'ordre $x_1 < x_2 < x_3$ si l'une des deux arêtes en pointillés est présente.

valeurs de D_{x_j} compatibles avec $a \in D_{x_i}$. Ainsi, s'il y a une contrainte dont la portée est $\{i, j\}$, alors $R(c_{ij})[a] = \{b \in D_{x_j} | (a, b) \in R(c_{ij})\}$; s'il n'y pas de contrainte dont la portée est $\{i, j\}$, alors, par défaut, $R(c_{ij})[a] = D_{x_j}$. Nous rappelons la propriété BTP introduite dans [1].

Définition (BTP) Une instance de CSP binaire (X, D, C) satisfait la *Broken Triangle Property* (BTP) par rapport à un ordre sur les variables $<$ si, pour tout triplet de variables (x_i, x_j, x_k) tel que $i < j < k$, si $(v_i, v_j) \in R(c_{ij})$, $(v_i, v_k) \in R(c_{ik})$ et $(v_j, v'_k) \in R(c_{jk})$, alors soit $(v_i, v'_k) \in R(c_{ik})$, soit $(v_j, v_k) \in R(c_{jk})$. Si aucun de ces deux couples n'existe, (v_i, v_j, v_k, v'_k) est appelé *triangle cassé sur x_k par rapport à x_i et x_j*.

S'il existe au moins un triangle cassé sur x_k par rapport à x_i et x_j, (x_i, x_j, x_k) est appelé *triplet cassé sur x_k par rapport à x_i and x_j*. Soit BTP l'ensemble des instances pour lesquelles il existe un ordre sur les variables tel que BTP est vérifiée par rapport à cet ordre. La propriété BTP est relative à la compatibilité entre valeurs des domaines et peut donc être représentée graphiquement (figure 1) à l'aide du graphe de microstructure. Par exemple, dans la figure 1 (a), il y a un triangle cassé sur x_3 par rapport aux variables x_1 et x_2 puisque nous avons $(v_1, v'_3) \notin R(c_{13})$ et $(v_2, v_3) \notin R(c_{23})$ tandis que $(v_1, v_2) \in R(c_{12})$, $(v_1, v_3) \in R(c_{13})$ et $(v_2, v'_3) \in R(c_{23})$ est vérifiée. Aussi, (x_1, x_2, x_3) est un triplet cassé sur x_3 par rapport à x_1 et x_2. Par contre, dans la figure 1 (b), si l'une des deux arêtes en pointillés (c'est-à-dire des 2-uplets) figure dans la microstructure, la propriété BTP sera vérifiée pour tous les ordres sur les variables.

Très récemment, un travail autour de la propriété BTP a conduit à proposer une propriété voisine, appelée *ETP* pour *Extendable-Triple Property* [13] qui est basée sur une relaxation des conditions de BTP, en considérant quatre variables plutôt que trois, et en tolérant l'existence de certains triangles cassés.

Définition (ETP) Une instance de CSP binaire (X, D, C) satisfait la *Extendable-Triple Property* (ETP) par rapport à un ordre sur les variables $<$ si et seulement si, pour tout quadruplet de variables (x_i, x_j, x_k, x_l) tel que $i < j < k < l$, il existe au plus un triplet cassé sur x_l parmi (x_i, x_j, x_l), (x_i, x_k, x_l) et (x_j, x_k, x_l).

De cette façon, une instance de CSP binaire peut satisfaire la propriété ETP quand bien même elle contiendrait deux triplets cassés parmi (x_i, x_j, x_k, x_l), un sur x_k, et un autre sur x_l, alors qu'aucun n'est autorisé avec BTP. Ainsi, ETP généralise strictement BTP puisqu'une instance peut satisfaire ETP en invalidant BTP alors que l'inverse est faux. Une conséquence immédiate est que la classe des instances satisfaisant BTP est strictement incluse dans la classe des instances satisfaisant ETP (notée ETP) comme indiqué dans le théorème 1 de [13]. i.e. $BTP \subsetneq ETP$. Comme dans le cas de BTP, ETP nous permet de définir une classe polynomiale mais pour cela, il faut imposer une propriété supplémentaire liée au niveau de cohérence locale qui doit être vérifiée par les instances. Alors que l'ensemble des instances satisfaisant BTP définit une classe polynomiale, l'ensemble des instances satisfaisant ETP requiert de plus la satisfaction de la *cohérence de chemin forte* (*Strong-Path-Consistency* [9]), c'est-à-dire la cohérence d'arc et la cohérence de chemin. Néanmoins, ces instances vont conserver certaines des propriétés intéressantes vérifiées par les instances satisfaisant BTP, comme par exemple leur capacité à être résolues en temps polynomial par des algorithmes usuels tels que MAC ou RFL. Dans la partie suivante, nous introduisons une nouvelle propriété qui généralise BTP mais également ETP.

3 k-BTP : définition et propriétées

Définition (k-BTP) Une instance de CSP binaire $P = (X, D, C)$ satisfait la propriété k-BTP pour un k donné $(2 \leq k < n)$ par rapport à un ordre sur les variables $<$ si et seulement si, pour tout sous-ensemble de $k + 1$ variables $x_{i_1}, x_{i_2}, \ldots x_{i_{k+1}}$ tel que $i_1 < i_2 < \ldots < i_{k-1} < i_k < i_{k+1}$, il existe au moins un triplet de variables $(x_{i_j}, x_{i_{j'}}, x_{i_{k+1}})$ avec $1 \leq j \neq j' \leq k$ tel qu'il n'existe pas de triangle cassé sur $x_{i_{k+1}}$ par rapport à x_{i_j} et $x_{i_{j'}}$.

On notera k-BTP l'ensemble des instances pour lesquelles il existe un ordre sur les variables tel que k-BTP est vérifiée par rapport à cet ordre. On peut constater que 2-BTP est exactement BTP alors que 3-BTP inclut ETP. À partir de là, on peut immédiatement étendre le théorème 1 de [13] puisque $BTP \subsetneq$

$ETP \subsetneq 3\text{-}BTP$. Mais au-delà, un résultat plus général, qui est une conséquence immédiate de la définition de k-BTP peut être formulé :

Théorème 1 *Pour tout $k \geq 2$, on a :*

$$k\text{-}BTP \subsetneq (k{+}1)\text{-}BTP$$

Pour analyser la polynomialité du traitement de k-BTP, nous montrons maintenant que les instances de cette classe peuvent être reconnues en temps polynomial :

Théorème 2 *Étant donnée une instance de CSP binaire (X, D, C) et une constante k avec $2 \leq k < n$, il existe un algorithme de complexité polynomiale pour déterminer un ordre sur les variables $<$ tel que cette instance satisfait k-BTP par rapport à $<$, ou pour déterminer qu'un tel ordre n'existe pas.*

Preuve : Comme dans la preuve correspondante pour BTP [1] et ETP [13], nous définissons une instance de CSP notée P_o qui est cohérente si et seulement si un ordre permettant de vérifier k-BTP existe. Plus précisément, cette instance possède une variable o_i de domaine $\{1, \ldots, n\}$ par variable x_i de X. La valeur de o_i représente la position de la variable x_i dans l'ordre. Nous ajoutons une contrainte concernant $\{o_{i_1}, o_{i_2}, \ldots o_{i_k}, o_{i_{k+1}}\}$ et imposant la condition $o_{i_{k+1}} < max(o_{i_1}, o_{i_2}, \ldots o_{i_k})$ pour chaque $k{+}1$-uplet de variables $(x_{i_1}, x_{i_2}, \ldots x_{i_k}, x_{i_{k+1}})$ tel que chaque triplet de variables $(x_{i_j}, x_{i_{j'}}, x_{i_{k+1}})$ avec $1 \leq j \neq j' \leq k$ possède au moins un triangle cassé sur $x_{i_{k+1}}$ par rapport à x_{i_j} et $x_{i_{j'}}$. Si P_o possède une solution, un ordre total $<$ sur les variables peut être produit à partir de l'ordre partiel donné par les valeurs des variables o_i. Alors, pour chaque $k{+}1$-uplet de variables $(x_{i_1}, x_{i_2}, \ldots x_{i_k}, x_{i_{k+1}})$, avec $i_1 < \ldots < i_{k+1}$, nous avons au moins un triplet de variables $(x_{i_j}, x_{i_{j'}}, x_{i_{k+1}})$ avec $1 \leq j \neq j' \leq k$ qui n'a pas de triangle cassé sur $x_{i_{k+1}}$ par rapport à x_{i_j} et $x_{i_{j'}}$. En effet, si ce n'était pas le cas, alors la contrainte $o_{i_{k+1}} < max(o_{i_1}, o_{i_2}, \ldots o_{i_k})$ aurait été imposée, ce qui serait en contradiction avec $i_1 < \ldots < i_{k+1}$. Donc, si P_o possède une solution, nous avons un ordre satisfaisant la propriété k-BTP.

Inversement, considérons un ordre satisfaisant la propriété k-BTP et supposons que P_o n'a pas de solution. Cela signifie qu'au moins une contrainte $o_{i_{k+1}} < max(o_{i_1}, o_{i_2}, \ldots o_{i_k})$ est violée. Donc chaque triplet de variables $(x_{i_j}, x_{i_{j'}}, x_{i_{k+1}})$ avec $1 \leq j \neq j' \leq k$ possède au moins un triangle cassé sur $x_{i_{k+1}}$, ce qui est impossible puisque cet ordre vérifie la propriété k-BTP. Ainsi P_o possède une solution si et seulement si (X, D, C) admet un ordre satisfaisant la propriété k-BTP. Nous montrons maintenant que P_o peut être construit et résolu en temps polynomial. Trouver tous les triplets cassés peut être réalisé en $O(n^3.d^4)$, tandis que la définition des contraintes $o_{i_{k+1}} < max(o_{i_1}, o_{i_2}, \ldots o_{i_k})$ peut être réalisée en $O(n^{k+1})$. Donc P_o peut être calculé en $O(n^3.d^4 + n^{k+1})$. En outre, P_o peut être résolu en temps polynomial en établissant la cohérence d'arc généralisée puisque ses contraintes sont *max-closed* [12]. □

Nous analysons maintenant la complexité de la résolution des instances de la classe $k\text{-}BTP$. Afin d'assurer la polynomialité de $k\text{-}BTP$, nous considérons une condition supplémentaire qui porte sur la vérification par les instances de la *k-cohérence-forte* dont nous rappelons la définition.

Definition (k-cohérence-forte [9]) *Une instance de CSP binaire satisfait la i-cohérence si toute affectation cohérente de $i-1$ variables peut être étendue à une affectation cohérente sur toute $i^{ème}$ variable. Une instance de CSP binaire satisfait la k-cohérence-forte si elle satisfait la i-cohérence pour tout i tel que $1 \leq i \leq k$.*

La k-cohérence-forte et $k\text{-}BTP$ permettent de définir une nouvelle classe polynomiale :

Théorème 3 *Soit P une instance de CSP binaire telle qu'il existe une constante k avec $2 \leq k < n$ pour laquelle P satisfait k-BTP par rapport à un ordre sur les variables $<$ ainsi que la k-cohérence-forte. Alors l'instance P est cohérente et une solution peut être trouvée en temps polynomial.*

Preuve : Considérons un ordre pour l'affectation des variables correspondant à l'ordre $<$. Comme l'instance satisfait la k-cohérence-forte, elle satisfait la cohérence d'arc et par conséquent, aucun de ses domaines n'est vide et chaque valeur possède au moins un support dans chacun des autres domaines. De plus, comme l'instance satisfait la k-cohérence-forte, il existe une affectation cohérente des k premières variables. Maintenant, et plus généralement, supposons que nous disposons d'une affectation cohérente $(u_1, u_2, \ldots u_{l-1}, u_l)$ pour les l premières variables $x_1, x_2, \ldots x_{l-1}, x_l$ dans l'ordre, avec $k \leq l < n$. Nous montrons que cette affectation peut être étendue de façon cohérente sur la variable x_{l+1}. Pour montrer cela, nous devons prouver que $\cap_{1 \leq i \leq l} R(c_{il+1})[u_i] \neq \emptyset$, c'est-à-dire qu'il existe au moins une valeur dans le domaine de x_{l+1} qui est compatible avec l'affectation $(u_1, u_2, \ldots u_{l-1}, u_l)$.

Nous prouvons d'abord cela pour $l = k$. Considérons l'affectation cohérente $(u_1, u_2, \ldots u_{k-1}, u_k)$ sur les k premières variables. Considérons une $k{+}1^{ème}$ variable

x_{k+1} apparaissant plus loin dans l'ordre. Puisque P satisfait k-BTP, il existe au moins un triplet de variables $(x_j, x_{j'}, x_{k+1})$ avec $1 \leq j \neq j' \leq k$ tel qu'il n'existe pas de triangle cassé sur x_{k+1} par rapport à x_j et $x_{j'}$. D'après le lemme 2.4 présenté dans [1], nous avons :

$$(R(c_{jk+1})[u_j] \subseteq R(c_{j'k+1})[u_{j'}])$$

ou

$$(R(c_{j'k+1})[u_{j'}] \subseteq R(c_{jk+1})[u_j])$$

Sans manque de généralité, supposons que $R(c_{jk+1})[u_j] \subseteq R(c_{j'k+1})[u_{j'}]$ et $j < j'$. Puisque P satisfait la k-cohérence-forte, nous savons que l'affectation partielle de $(u_1, u_2, \ldots, u_j, \ldots u_{k-1}, u_k)$ sur $k-1$ variables excluant l'affectation $u_{j'}$ pour $x_{j'}$ peut être étendue de façon cohérente sur la variable x_{k+1}. De plus, nous savons que $R(c_{jk+1})[u_j] \subseteq R(c_{j'k+1})[u_{j'}]$ et par la cohérence d'arc, $R(c_{i_j i_{k+1}})[u_j] \neq \emptyset$. Par conséquent, $(u_1, u_2, \ldots, u_j, \ \ldots, \ u_{j'}, \ \ldots, u_k, u_{k+1})$ est une affectation cohérente des $k+1$ premières variables.

Notons que cette preuve est également valide pour tous les sous-ensembles de $k+1$ variables tels que x_{k+1} apparaît plus loin dans l'ordre $<$, et pas seulement pour les $k+1$ premières variables $x_1, x_2, \ldots x_{k-1}, x_k$ et x_{k+1}.

Maintenant, nous démontrons la propriété pour l avec $k < l < n$. C'est-à-dire que nous montrons qu'une affectation cohérente $(u_1, u_2, \ldots u_{l-1}, u_l)$ peut être étendue à une $(l+1)^{ème}$ variable. Comme hypothèse d'induction, nous supposons que chaque affectation cohérente sur $l-1$ variables peut être étendue à une $l^{ème}$ variable qui apparaît plus loin dans l'ordre considéré.

Considérons une affectation cohérente $(u_1, u_2, \ldots u_{l-1}, u_l)$ sur les l premières variables. Soit $(u_{i_1}, u_{i_2}, \ldots u_{i_k})$ une affectation partielle de $(u_1, u_2, \ldots u_{l-1}, \ u_l)$ sur k variables. Comme P satisfait k-BTP, et comme $k < l < n$, pour tous les sous-ensembles de k variables $x_{i_1}, x_{i_2}, \ldots x_{i_k}$, nous savons qu'il existe un triangle qui n'est pas cassé sur x_{l+1} par rapport à x_{i_j} et $x_{i_{j'}}$, avec x_{i_j} et $x_{i_{j'}}$ apparaissant dans les variables $x_{i_1}, x_{i_2}, \ldots x_{i_k}$. Aussi, sans manque de généralité, nous pouvons considérer que $i_1 \leq i_j < i_{j'} \leq i_k \leq l$ et que nous avons $R(c_{i_j l+1})[u_{i_j}] \subseteq R(c_{i_{j'} l+1})[u_{i_{j'}}]$. Notons que x_{i_j} et $x_{i_{j'}}$ peuvent être permutées dans l'ordre si c'est nécessaire.

Maintenant, considérons l'affectation cohérente $(u_1, u_2, \ldots u_{l-1}, u_l)$ sur les l premières variables. Par hypothèse d'induction, chaque affectation partielle de $(u_1, u_2, \ldots u_{l-1}, u_l)$ sur $l-1$ variables peut être étendue à une affectation cohérente sur x_{l+1} avec une valeur compatible u_{l+1}. Maintenant, considérons l'affectation partielle sur $l-1$ variables où $u_{i_{j'}}$ n'apparaît pas. Cette affectation est par exemple $(u_1, u_2, \ldots u_{i_j}, \ldots u_{l-1}, u_l, u_{l+1})$. Comme nous avons $R(c_{i_j l+1})[u_{i_j}] \subseteq R(c_{i_{j'} l+1})[u_{i_{j'}}]$, la valeur $u_{i_{j'}}$ est également compatible avec u_{l+1}, et donc l'affectation $(u_1, u_2, \ldots u_{i_j}, \ldots u_{i_{j'}}, \ldots u_{l-1}, u_l, u_{l+1})$ sur les $l+1$ premières variables est une affectation cohérente.

Ainsi, toute affectation cohérente $(u_1, u_2, \ldots u_{l-1}, u_l)$ sur $(x_1, x_2, \ldots x_{l-1}, x_l)$ peut être étendue à une $(l+1)^{ème}$ variable, pour tout l avec $k < l < n$. Et plus généralement, nous avons démontré que toute affectation cohérente sur l variables, pas nécessairement consécutives dans l'ordre (comme le sont les l premières variables), peut être étendue à une affectation cohérente sur toute $(l+1)^{ème}$ variable qui apparaît après ces l variables dans l'ordre $<$ associé à k-BTP. Ainsi, l'hypothèse d'induction est vérifiée pour l'étape suivante.

Notons que cette preuve démontre également qu'une instance qui satisfait la k-cohérence-forte et k-BTP par rapport à un ordre $<$ est cohérente.

Finalement, étant donné un ordre $<$, nous montrons que trouver une solution peut être réalisé en temps polynomial. Étant donnée une affectation cohérente $(u_1, u_2, \ldots u_l)$ avec $l < n$, trouver une valeur compatible u_{l+1} pour la variable suivante x_{l+1} est réalisable en recherchant dans son domaine une valeur compatible, sachant que la taille du domaine est majorée par d. Pour chaque valeur, nous devons vérifier les contraintes connectant la variable x_{l+1} ce qui peut être réalisé en $O(e_{l+1})$ si la variable suivante x_{l+1} possède e_{l+1} voisins parmi les variables précédentes. Puisque $\Sigma_{1 \leq l < n} e_{l+1} = e$, le coût total pour trouver une solution est $O((n+e).d)$. □

Dans la suite, nous noterons k-BTP-SkC, la classe des instances satisfaisant k-BTP et vérifiant la k-cohérence-forte. Une des propriétés les plus intéressantes de la classe BTP est le fait que les instances de cette classe peuvent être résolues en temps polynomial en utilisant des algorithmes classiques (tels que MAC ou RFL), cela mêmes qui sont mis en œuvre dans la plupart des solveurs. La propriété suivante établit un résultat similaire pour k-BTP-SkC. En effet, la preuve du théorème 3 nous permet de montrer que des algorithmes tels que BT (Backtracking), MAC et RFL peuvent résoudre (i.e. trouver une solution) toute instance de la classe k-BTP-SkC en temps polynomial :

Théorème 4 *Étant donné une instance P de CSP binaire et un ordre sur les variables $<$ tel que P satisfait k-BTP par rapport à $<$ et vérifie la k-cohérence-forte, les algorithmes BT, MAC et RFL trouvent une solution de l'instance P en temps polynomial.*

Preuve : Comme l'instance satisfait la k-cohérence-forte, BT utilisant l'ordre $<$ pour l'affectation des

variables peut trouver une affectation cohérente sur $x_1, x_2, \ldots x_{k-1}$ et x_k. De plus, étant donné l avec $k < l < n$, il est montré dans la preuve du théorème 3 qu'une affectation cohérente $(u_1, u_2, \ldots u_{l-1}, u_l)$ sur $x_1, x_2, \ldots x_{l-1}$ et x_l peut être étendue à une $(l+1)^{ème}$ variable, c'est-à-dire sur x_{l+1}. Pour trouver une affectation de x_{l+1}, nous devons chercher une valeur compatible dans son domaine. Cela est réalisable en $O(e_{l+1}.d)$ en supposant que x_{l+1} possède e_{l+1} voisins dans les variables précédentes. Donc, comme pour la preuve du théorème 3, trouver une solution de P est globalement réalisable en $O((n+e).d)$. Si nous considérons maintenant des algorithmes tels que MAC ou RFL, par le même raisonnement, nous montrons que leur complexité est limitée à $O(n.(n+e).d^2)$ en raison du coût supplémentaire du filtrage par cohérence d'arc effectué après chaque affectation de variable. □

Dans la partie 5, nous discutons de l'intérêt de la classe k-BTP d'un point de vue pratique. Avant cela, dans la partie suivante, nous étudions les relations existant entre k-BTP et certaines classes polynomiales de la littérature.

4 Relations existant entre k-BTP et d'autres classes polynomiales

Nous considérons tout d'abord une classe polynomiale très importante, à la fois dans le cadre des CSP, mais bien au-delà, la classe basée sur la notion de décomposition arborescente de graphes [16].

Définition (Décomposition arborescente) *Étant donné un graphe $G = (X, C)$, une* décomposition arborescente *de G est une paire (E, T) où $T = (I, F)$ est un arbre et $E = \{E_i : i \in I\}$ une famille de sous-ensembles (appelés* clusters*) de X, telle que chaque cluster E_i est un nœud de T et vérifie :*

(i) $\cup_{i \in I} E_i = X$,

(ii) pour chaque arête $\{x, y\} \in C$, il existe $i \in I$ avec $\{x, y\} \subseteq E_i$, et

(iii) pour tout $i, j, k \in I$, si k est sur un chemin de i vers j dans T, alors $E_i \cap E_j \subseteq E_k$.

La largeur d'une décomposition arborescente (E, T) est égale à $max_{i \in I}|E_i| - 1$. La largeur arborescente *ou* tree-width *w de G est la largeur minimale pour toutes les décompositions arborescentes de G.*

Soit k-TW la classe des instances de CSP binaires tels que leur largeur arborescente est inférieure ou égale à une constante k. Il est bien connu que k-TW constitue une classe polynomiale [11]. Récemment, M. Vardi a posé une question sur les relations qui pourraient exister entre k-TW et ETP ou d'autres généralisations de BTP [18]. Les deux théorèmes qui suivent donnent une première réponse à cette question.

Théorème 5 *k-TW $\subsetneq$ $(k+1)$-BTP.*

Preuve : Nous montrons tout d'abord que k-TW $\subseteq$ $(k+1)$-BTP. Il est bien connu que si la largeur arborescente d'une instance de CSP binaire est bornée par k, il existe un ordre $<$ sur les variables, tel que pour $x_i \in X$, $|\{x_j \in X : j < i \text{ et } c_{ji} \in C\}| \leq k$ [6]. Maintenant, considérons un sous-ensemble de $k+2$ variables $x_{i_1}, x_{i_2}, \ldots x_{i_k}, x_{i_{k+1}}, x_{i_{k+2}}$ tel que $i_1 < i_2 < \ldots < i_{k-1} < i_k < i_{k+1} < i_{k+2}$. Puisque la largeur arborescente est bornée par k, nous savons qu'il existe au plus k contraintes $c_{i_j i_{k+2}} \in C$. Donc, il y a au moins un triplet de variables $(x_{i_j}, x_{i_{j'}}, x_{i_{k+2}})$ avec $1 \leq j \neq j' \leq k$ tel que $c_{i_j i_{k+2}} \notin C$ ou $c_{i_{j'} i_{k+2}} \notin C$. Sans manque de généralité, supposons qu'il n'y a pas de contrainte $c_{i_j i_{k+2}} \in C$. Ainsi, il n'y a aucun triangle cassé sur $x_{i_{k+2}}$ par rapport à x_{i_j} et $x_{i_{j'}}$ parce que toutes les valeurs de $D_{x_{i_j}}$ sont compatibles avec toutes les valeurs de $D_{x_{i_{k+2}}}$. Ainsi, l'instance de CSP considérée satisfait la propriété $(k+1)$-BTP. Finalement, il est facile de définir des instances dont la largeur arborescente est strictement supérieure à k et qui satisfont la propriété $(k+1)$-BTP. Par exemple, nous pouvons considérer une instance de CSP monovalente (i.e. dont la taille des domaines vaut un) et dont le graphe de contraintes est complet, et possédant une solution. La largeur arborescente de cette instance est $n-1$ alors qu'elle satisfait k-BTP pour toutes les valeurs possibles de k. □

Le coût de la vérification de cohérence d'instances de k-TW est du même ordre que celui de l'obtention de la $(k+1)$-cohérence-forte, qui est $O(n^{k+1}d^{k+1})$. Néanmoins, cela ne nous permet pas d'établir une inclusion formelle de k-TW dans $(k+1)$-BTP-$S(k+1)C$ qui est une classe polynomiale tandis $(k+1)$-BTP n'est pas nécessairement une classe polynomiale. Mais si l'on note k-TW-$S(k+1)C$, la classe des instances de CSP binaires appartenant à k-TW et qui satisfont la $(k+1)$-cohérence-forte, nous obtenons directement le résultat suivant :

Théorème 6 *Pour toute valeur $k < n$, on a :*

$$k\text{-}TW\text{-}S(k+1)C \subsetneq (k+1)\text{-}BTP\text{-}S(k+1)C.$$

Récemment, la classe polynomiale BTP a également été généralisée avec la définition de la propriété $\forall\exists$-BTP [2], d'une manière différente de celle que nous proposons ici, mais également en remarquant que tous les triangles cassés n'ont pas besoin d'être interdits.

Nous allons montrer que ces deux généralisations sont orthogonales.

Définition (∀∃-BTP) *Une instance de CSP binaire P satisfait la propriété* ∀∃-BTP *par rapport à un ordre des variables $<$ si et seulement si, pour chaque paire de variables x_i, x_k telles que $i < k$, pour tout valeur $v_i \in D_{x_i}$, $\exists v_k \in D_{x_k}$ telle que $(v_i, v_k) \in R(c_{ik})$ et pour toute variable x_j avec $j < k$ et $j \neq i$, et pour toute valeur $v_j \in D_{x_j}$ et pour toute valeur $v'_k \in D_{x_k}$, (v_i, v_j, v_k, v'_k) n'est pas un triangle cassé sur x_k par rapport à x_i et x_j. Soit ∀∃-BTP l'ensemble des instances pour lesquelles* ∀∃-BTP *est vérifiée par rapport à un certain ordre sur les variables.*

La classe ∀∃-BTP peut être résolue et reconnue en temps polynomial [2]. Elle constitue une classe polynomiale qui inclut strictement BTP car elle n'interdit pas tous les triangles cassés. Puisque k-BTP n'interdit également pas tous les triangles cassés, il est naturel de comparer ces deux classes. Nous analysons cela pour le cas particulier $k = 3$, mais le même argument vaut pour toute valeur de $k \geq 3$.

Théorème 7 *Même pour les ensembles d'instances de CSP binaires qui vérifient la cohérence de chemin forte, les propriétés* 3-BTP *et* ∀∃-BTP *sont incomparables.*

Preuve : Prenons une instance P^* pour laquelle chaque domaine D_{x_k} contient une valeur a^* telle que pour toutes les autres variables x_i, pour toutes les valeurs $v_i \in D_{x_i}$, $(v_i, a^*) \in R(c_{ik})$. On remarque que P^* satisfait ∀∃-BTP puisqu'il ne peut y avoir aucun triangle cassé de la forme (v_i, v_j, a^*, v'_k), la valeur a^* étant compatible avec toutes les affectations à toutes les autres variables. Il est facile de compléter une tellle instance P^* de sorte qu'elle ne satisfasse pas 3-BTP pour tout ordre sur les variables en rajoutant des triangles cassés sur d'autres valeurs de domaine que a^*.

Considérons une instance à trois variables notée P_3, avec des domaines $\{0, \ldots, 7\}$ et qui possède les trois contraintes suivantes :

$$\begin{aligned} x_1 = x_2 \quad &\wedge \quad x_1 \equiv x_2 + 1 \pmod 8 \\ x_2 = x_3 \quad &\wedge \quad x_2 \equiv x_3 + 2 \pmod 8 \\ x_1 = x_3 \quad &\wedge \quad x_1 \equiv x_3 + 4 \pmod 8 \end{aligned}$$

Ainsi, P_3 vérifie la cohérence de chemin forte et satisfait trivialement 3-BTP (car il n'y a que trois variables), mais P_3 ne satisfait pas ∀∃-BTP, quel que soit l'ordre des variables. □

Nous considérons maintenant une classe polynomiale très générale récemment découverte par Naanaa [14] et qui mérite sans aucun doute d'être mieux connue.

Soit E un ensemble fini et soit $\{E_i\}_{i \in I}$ une famille finie de sous-ensembles de E. La famille $\{E_i\}_{i \in I}$ est dite *indépendante* si et seulement si pour tout $J \subset I$,

$$\bigcap_{i \in I} E_i \ \subset \ \bigcap_{j \in J} E_j.$$

Notons que $\{E_i\}_{i \in I}$ ne peut être indépendante si $\exists j \neq j' \in I$ tels que $E_j \subseteq E_{j'}$ puisque dans ce cas et avec $J = I \setminus \{j'\}$, nous devrions avoir

$$\bigcap_{i \in I} E_i \ = \ \bigcap_{j \in J} E_j.$$

Définition (Rang Directionnel) *Soit P une instance de CSP binaire dont les variables sont totalement ordonnées par $<$. Le* rang directionnel *de la variable x_m est la taille k de la plus grande affectation cohérente $(a_1, \ldots, a_k)$ à un ensemble de variables $x_{i_1}, \ldots, x_{i_k}$ (avec $i_1 < \ldots < i_k < m$) telle que la famille des ensembles $\{R(c_{i_j m})[a_j]\}_{j=1,\ldots,k}$ est indépendante. Le rang directionnel de P (par rapport à l'ordre $<$ de ses variables) est le rang directionnel maximal sur toutes ses variables.*

Naanaa a montré que si P est une instance de CSP binaire qui a un rang directionnel ne dépassant pas k et s'il vérifie la $(k+1)$-cohérence-forte, alors P est globalement cohérente [14]. On note DR-k, l'ensemble de ces instances. Naanaa souligne que certaines classes polynomiales connues telles que les instances de CSP binaires avec des contraintes *connected row convex* [7], ont un rang directionnel borné.

Si une instance de CSP binaire P est $(k+1)$-BTP, alors aucune variable peut avoir un rang directionnel plus grand que k. Ceci est dû au fait que pour toute variable x_m et toute affectation $(a_1, \ldots, a_{k+1})$ d'un ensemble de variables $x_{i_1}, \ldots, x_{i_{k+1}}$ avec $i_1 < \ldots < i_{k+1} < m$, par la définition de $(k+1)$-BTP, nous devons avoir $R(c_{i_j m})[a_j] \subseteq R(c_{i_{j'} m})[a_{j'}]$ pour certains $j \neq j' \in \{1, \ldots, k+1\}$. Donc, comme observé ci-dessus, les ensembles $\{R(c_{i_j m})[a_j]\}_{j=1,\ldots,k+1}$ ne peuvent pas être indépendants. Il en résulte que la polynomialité de $(k+1)$-BTP-$S(k+1)C$ est également un corollaire du résultat de Naanaa [14]. Toutefois, la propriété $(k+1)$-BTP, bien que subsumée par DR-k, peut être détectée en $O(n^k d^k + n^3 d^4)$ ce qui inférieur comparativement à $O(n^{k+1} d^{k+1})$ pour DR-k.

5 Expérimentations

Dans cette section, nous comparons l'intérêt pratique des classes polynomiales BTP, ETP-SPC, k-BTP-SkC et DR-(k-1), où SPC notera la cohérence

de chemin forte, soit la 3-cohérence-forte. Nous ne considérons que les cas pour lesquels $k = 3$, puisque l'établissement de la k-cohérence-forte devient trop coûteuse en temps, mais aussi du fait que pour $k > 3$, cela peut conduire à l'ajout de contraintes d'arité $k-1$. Nos expériences concernent 2 373 instances binaires de la troisième compétition de solveurs CSP [1]. Il s'agit donc d'instances de CSP binaires issues des benchmarks utilisés pour les évaluations et les comparaisons de solveurs. Nous mettons d'abord en évidence l'existence d'instances appartenant à certaines des classes polynomiales considérées ici. Ce sont les mêmes benchmarks qui ont été utilisés dans [13]. Ensuite, nous évaluons l'impact de ces classes polynomiales sur l'efficacité des solveurs. La plupart des solveurs de l'état de l'art reposant sur les algorithmes MAC ou RFL, nous présentons donc ici les résultats obtenus sur MAC mais on peut noter que nous avons observé des résultats similaires avec RFL.

Puisque les classes polynomiales $ETP\text{-}SPC$, 3-$BTP\text{-}SPC$ et DR-2 exigent la vérification de la cohérence de chemin forte, nous devons d'abord réaliser un tel filtrage sur chaque instance avant de vérifier si elle appartient aux classes considérées, ceci dans le même esprit que [8]. Ce faisant, 628 instances ont été détectées comme incohérentes et donc elles appartiennent trivialement à toutes ces classes polynomiales. 85 des instances restantes appartiennent à 3-$BTP\text{-}SPC$, alors que 87 ont un rang directionnel inférieur ou égal à deux. Parmi ces instances, nous avons respectivement 71 et 76 instances appartenant à $BTP\text{-}SPC$ et à $ETP\text{-}SPC$. Les différences entre ces classes polynomiales sont bien mises en évidence par certaines instances de la famille bqwh-15-106 puisque nous pouvons observer toutes les configurations possibles des relations d'inclusions $BTP\text{-}SPC \subsetneq ETP\text{-}SPC \subsetneq$ 3-$BTP\text{-}SPC \subsetneq DR$-2. Par exemple, l'instance bqwh-15-106-13 appartient à toutes les classes polynomiales considérées alors que les instances bqwh-15-106-28, bqwh-15-106-16 et bqwh-15-106-76 appartiennent seulement respectivement à trois, deux ou une de ces classes polynomiales. Le tableau 5 présente certaines instances appartenant aux classes $ETP\text{-}SPC$, 3-$BTP\text{-}SPC$ ou DR-2. Ce tableau fournit également la largeur arborescente w de ces instances et leur largeur arborescente w' une fois que le filtrage par cohérence de chemin forte a été appliqué. Quand la largeur arborescente exacte est inconnue (rappelons que le calcul d'une décomposition arborescente optimale est un problème NP-difficile), nous donnons son encadrement par un intervalle. Nous pouvons noter la diversité de ces instances (instances académiques, aléatoires ou issues du monde réel). Certaines de ces instances appartiennent à 3-$BTP\text{-}SPC$ ou DR-2 grâce à leur structure. Par exemple, graph12-w0 et hanoi-7 ont un graphe de contraintes acyclique tandis que la largeur arborescente de domino-100-100 et crossword-m1-uk-puzzle01 est de deux. Cependant, la plupart des instances ont une largeur arborescente supérieure à deux. En outre, dans la plupart des cas, l'application de SPC peut augmenter de manière significative la largeur arborescente d'origine de ces instances. Par exemple, la largeur arborescente de l'instance driverlogw-09-sat est initialement majorée par 108 et est égale à 629 après l'application du filtrage SPC. Cette augmentation est expliquée par les paires de valeurs qui sont interdites par SPC et dont la mise en évidence conduit à la représentation de contraintes induites dont l'ajout a pour effet de densifier le graphe de contraintes. Lorsque SPC interdit une paire de valeurs (v_i, v_j) pour une paire donnée de variables (x_i, x_j), le filtrage supprime (v_i, v_j) de la relation $R(c_{ij})$ si la contrainte c_{ij} existe. Cependant, si la contrainte c_{ij} n'existe pas encore, le filtrage SPC doit d'abord l'ajouter au problème. Dans un tel cas, du fait des contraintes supplémentaires et de leur nombre, la largeur arborescente peut augmenter de manière significative. Il est à noter que pour les instances considérées dont la largeur arborescente est d'au plus deux initialement, la largeur arborescente demeure inchangée après l'application du filtrage SPC.

En ce qui concerne la résolution, toutes les instances appartenant à 3-$BTP\text{-}SPC$ ou DR-2 sont résolues par MAC sans aucun retour arrière, sauf l'instance driverlogw-04c-sat qui n'y a recours qu'une seule fois. Il faut noter que MAC n'a pas connaissance de l'ordre sur les variables nécessaire à la satisfaction de 3-BTP ni d'ailleurs de l'ordre associé à un rang directionnel au plus égal à deux. Dans la plupart des cas, nous avons d'ailleurs observé que l'instance de CSP utilisée dans la preuve du théorème 2 afin de calculer un ordre sur les variables approprié n'a pas de contrainte. Ainsi, tout ordre des variables est approprié. En revanche, pour une douzaine d'instances, ce CSP possède plusieurs contraintes mais reste nettement sous-contraint et le réseau de contraintes correspondant possède plusieurs composantes connexes. Il en résulte que le CSP associé à l'ordre possède généralement un grand nombre de solutions. Il est donc très probable que MAC exploite implicitement pour l'affectation des variables un ordre approprié. Par exemple, le CSP associé à l'ordre pour vérifier si l'instance bqwh-15-106-76 (qui possède 106 variables) a un rang directionnel majoré par deux possède 65 composantes connexes et admet plus de 33 millions de solutions.

Certaines des instances sont résolues efficacement par MAC sans retour arrière, même si elles ne font

1. Voir http ://www.cril.univ-artois.fr/CPAI08.

Instance	n	w	w'	BTP-SPC	ETP-SPC	3-BTP-SPC	DR-2
bqwh-15-106-13	106	$[7, 48]$	104	oui	oui	oui	oui
bqwh-15-106-16	106	$[6, 45]$	99	non	non	oui	oui
bqwh-15-106-28	106	$[7, 52]$	105	non	oui	oui	oui
bqwh-15-106-76	106	$[6, 44]$	100	non	non	non	oui
bqwh-15-106-77	106	$[7, 50]$	100	non	non	oui	oui
bqwh-18-141-33	141	$[7, 64]$	134	oui	oui	oui	oui
bqwh-18-141-57	141	$[7, 66]$	137	oui	oui	oui	oui
domino-100-100	100	2	2	oui	oui	oui	oui
domino-5000-500	5000	2	2	oui	oui	oui	oui
driverlogw-04c-sat	272	$[19, 56]$	$[214, 221]$	non	non	non	oui
driverlogw-09-sat	650	$[39, 108]$	629	oui	oui	oui	oui
fapp17-0300-10	300	$[6, 153]$	$[6, 154]$	oui	oui	oui	oui
fapp18-0350-10	350	$[5, 192]$	$[12, 199]$	oui	oui	oui	oui
fapp23-1800-9	1800	$[6, 1325]$	$[41, 1341]$	oui	oui	oui	oui
graph12-w0	680	1	1	oui	oui	oui	oui
graph13-w0	916	1	1	oui	oui	oui	oui
hanoi-7	126	1	1	oui	oui	oui	oui
langford-2-4	8	7	7	oui	oui	oui	oui
lard-83-83	83	82	82	non	non	oui	oui
lard-91-91	91	90	90	non	non	oui	oui
os-taillard-4-100-0	16	$[3, 9]$	15	oui	oui	oui	oui
os-taillard-4-100-9	16	$[3, 9]$	15	oui	oui	oui	oui
scen5	400	$[11, 32]$	$[167, 188]$	non	non	oui	oui

TABLE 1 – Quelques instances appartenant à BTP-SPC, ETP-SPC, 3-BTP-SPC ou DR-2 après l'application de SPC, avec leur largeur arborescente w et la largeur arborescente w' des instances une fois SPC appliquée.

pas partie de l'une des classes polynomiales étudiées. Aussi, nous considérons maintenant la notion de *backdoor* [19] avec l'objectif de proposer des explications au sujet de cette efficacité, dans le même esprit que dans [13]. Un *backdoor* est un ensemble de variables défini par rapport à une classe polynomiale, tel que, une fois ces variables affectées, le sous-problème induit se situe dans cette classe. Ici, nous nous intéressons à des backdoors qui sont découverts implicitement par MAC lors de l'affectation de certaines variables. En effet, après quelques affectations et l'application du filtrage, la partie restante du problème peut devenir finalement polynomiale. Afin de tenter d'observer ce phénomène, nous allons évaluer le nombre de variables qui doivent être affectées avant que MAC ne trouve implicitement un backdoor défini par rapport à l'une des classes étudiées ici. Sur les 50 instances examinées ici, nous avons observé que MAC trouve un backdoor relatif à BTP après avoir affecté plus de variables que pour les autres classes considérées. Les nombres de variables affectées nécessaires pour trouver un backdoor respectivement pour ETP et 3-BTP sont très proches, voire égaux dans la plupart des cas. En considérant DR-2, nous gagnons quelques variables par rapport à ETP et 3-BTP. Par exemple, MAC a besoin d'affecter au plus cinq variables avant de trouver un backdoor par rapport à 3-BTP ou DR-2 pour 14 instances contre 12 et 4 instances pour ETP et BTP [2]. Bien sûr, les instances qui en résultent ne satisfont pas nécessairement la cohérence de chemin forte et cela ne nous permet pas d'exploiter le théorème 4 pour expliquer l'efficacité de MAC. Néanmoins, comme évoqué ci-dessus, le grand nombre de solutions du CSP correspondant à l'existence de "bons ordres" permet probablement à MAC d'exploiter implicitement un ordre approprié.

6 Conclusion

Cet article présente une nouvelle famille de classes polynomiales pour les CSP binaires, k-BTP, et dont la polynomialité est associée à un niveau donné de k-cohérence-forte. Cette famille est basée sur une hiérarchie de classes d'instances dont la classe BTP constitue le cas de base (BTP = 2-BTP). Alors que BTP est définie sur des sous-ensembles de 3 variables, les classes k-BTP sont définies sur des ensembles de k+1 variables, tout en relaxant les conditions restrictives imposées par BTP. Nous avons montré que k-BTP hérite de certaines des propriétés de BTP, comme par

2. Notons que ces instances ne comprennent pas toutes celles mentionnées dans [13] puisque certaines d'entre elles appartiennent déjà à 3-BTP-SPC et/ou DR-2.

exemple la possiblité d'être résolue en temps polynomial en utilisant des algorithmes standards tels que MAC. Nous avons également démontré que k-BTP généralise strictement la classe des instances dont la largeur arborescente est bornée par une constante et nous avons analysé les relations avec la classe basée sur la notion de *rang directionnel* récemment introduite par Naanaa. Pour évaluer l'intérêt pratique de la classe k-BTP, une analyse expérimentale est présentée. Elle se concentre sur le cas particulier de 3-BTP. Cette analyse montre un avantage significatif de k-BTP, comparativement à BTP et aux CSP de largeur arborescente bornée.

Un développement de ces travaux semble nécessaire pour déterminer si la condition correspondant à la k-cohérence-forte est réellement nécessaire ou si une condition plus faible suffirait pour garantir la polynomialité. En effet, les expériences ont montré que MAC peut résoudre sans retour arrière certaines instances appartenant à 3-BTP même lorsque ces instances ne vérifient pas le niveau correspondant de cohérence. D'un point de vue théorique et pratique, un défi intéressant porte sur la recherche du niveau minimum de cohérence requis parmi les différents types de cohérences locales telles que PIC [10], maxRPC [4] ou SAC [5]. En outre, l'étude d'une relaxation de la condition k-BTP devrait être abordée de manière à étendre la classe des instances qui peuvent être résolues en temps polynomial, mais dans une direction différente de celle proposée dans [14], même si d'autres développements théoriques et expérimentaux semblent clairement nécessaires pour véritablement apprécier toutes les conséquences du résultat de Naanaa. Enfin, il pourrait être intéressant d'étudier, déjà sur le cas particulier de 3-BTP, une approche similaire à celle introduite dans [3] où une nouvelle opération de réduction réalisable en temps polynomial basée sur la fusion des valeurs de domaines est proposée.

Références

[1] M. Cooper, Peter Jeavons, and Andras Salamon. Generalizing constraint satisfaction on trees : hybrid tractability and variable elimination. *Artificial Intelligence*, 174 :570–584, 2010.

[2] Martin C. Cooper. Beyond consistency and substitutability. In *Proceedings of CP*, pages 256–271, 2014.

[3] Martin C. Cooper, Achref El Mouelhi, Cyril Terrioux, and Bruno Zanuttini. On broken triangles. In *Proceedings of CP*, pages 9–24, 2014.

[4] R. Debruyne and C. Bessière. From restricted path consistency to max-restricted path consistency. In *Proceedings of CP*, pages 312–326, 1997.

[5] R. Debruyne and C. Bessière. Domain Filtering Consistencies. *Journal of Artificial Intelligence Research*, 14 :205–230, 2001.

[6] R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38 :353–366, 1989.

[7] Yves Deville, Olivier Barette, and Pascal Van Hentenryck. Constraint satisfaction over connected row convex constraints. *Artificial Intelligence*, 109(1-2) :243–271, 1999.

[8] A. El Mouelhi, P. Jégou, and C. Terrioux. Hidden Tractable Classes : from Theory to Practice. In *Proceedings of ICTAI*, pages 437–445, 2014.

[9] E. Freuder. A Sufficient Condition for Backtrack-Free Search. *Journal of the ACM*, 29 (1) :24–32, 1982.

[10] E. Freuder and C.D. Elfe. Neighborhood inverse consistency preprocessing. In *Proceedings of AAAI*, pages 202–208, 1996.

[11] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :343–282, 2000.

[12] P. Jeavons and M. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2) :327–339, 1995.

[13] P. Jégou and C. Terrioux. The Extendable-Triple Property : a new CSP Tractable Class beyond BTP. In *Proceedings of AAAI*, 2015.

[14] Wady Naanaa. Unifying and extending hybrid tractable classes of csps. *Journal of Experimental and Theoretical Artificial Intelligence*, 25(4) :407–424, 2013.

[15] B. Nadel. *Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms*, pages 287–342. In *Search in Artificial Intelligence*. Springer-Verlag, 1988.

[16] N. Robertson and P.D. Seymour. Graph minors II : Algorithmic aspects of treewidth. *Algorithms*, 7 :309–322, 1986.

[17] D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of ECAI*, pages 125–129, 1994.

[18] Moshe Vardi. Private communication. January 2015.

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

Clustering avec la minimisation de la somme des carrés par la programmation par contraintes

Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain

Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067, Orléans, France
{thi-bich-hanh.dao, khanh-chuong.duong, christel.vrain}@univ-orleans.fr

Résumé

Le clustering sous contraintes utilisateur a connu un essor important en fouille de données. Dans les dix dernières années, beaucoup de travaux se sont attachés à étendre les algorithmes classiques pour prendre en compte des contraintes utilisateur, mais ils sont en général limités à un seul type de contraintes. Dans de précédents travaux, nous avons proposé un cadre générique et déclaratif, fondé sur la programmation par contraintes, qui permet de modéliser différentes tâches de clustering sous contraintes. L'utilisateur peut spécifier un parmi plusieurs critères d'optimisation et combiner différents types de contraintes. Dans ce papier nous étendons le modèle pour traiter le critère le plus connu de clustering, qui est la minimisation de la somme des distances au carré des objets au centre de leur cluster. C'est un problème difficile et à notre connaissance, il existe une seule méthode exacte permettant d'intégrer des contraintes utilisateurs ; elle est fondée sur la programmation linéaire sur les entiers et la génération de colonnes. Nous développons un algorithme de filtrage pour la nouvelle contrainte spécifiant ce critère. Des expérimentations sur des bases de données classiques montrent que notre modèle obtient une meilleure performance que la méthode exacte fondée sur la programmation linéaire.

1 Introduction

La classification non supervisée (souvent appelée par le terme anglais "clustering") est une tâche importante en fouille de données. Elle vise à regrouper les données en un ensemble de classes ou clusters, de façon à ce que les objets d'une même classe aient une forte similarité entre eux et diffèrent fortement des objets des autres classes. Elle est souvent représentée par un problème d'optimisation. Différents critères d'optimisation existent, comme par exemple minimiser le diamètre maximal des clusters, ou maximiser la marge minimale entre clusters. Depuis les années 2000, des contraintes utilisateur sont introduites dans la tâche de clustering pour guider la recherche et pour atteindre les solutions souhaitées s'il en existe. L'extension avec des contraintes utilisateur se fait soit en adaptant les algorithmes classiques pour gérer les contraintes, soit en modifiant la distance entre données pour traduire les contraintes. Cependant, la plupart des approches ne garantissent pas la satisfaction de toutes le contraintes ou la qualité de la solution. De plus, chaque algorithme est spécifique pour un critère. Dans de précédents travaux [6, 7, 8], nous proposons un cadre général basé sur la programmation par contraintes pour modéliser le clustering sous contraintes utilisateur. Ce cadre intègre différents critères : minimiser le diamètre maximal des clusters, maximiser la marge minimale entre clusters ou minimiser la somme des dissimilarités intra-cluster. Différents types de contraintes utilisateurs sont également intégrés. Le cadre permet de trouver une solution qui est un optimum global et qui satisfait toutes les contraintes utilisateur, s'il en existe une. Dans ce papier, nous développons ce cadre pour intégrer le critère de minimisation de la somme des carrés WCSS (Within-Cluster Sum of Squares), qui est défini par la somme des distances au carré des objets de chaque cluster au centre du cluster.

Ce critère est le plus souvent utilisé dans les tâches de clustering pour définir des clusters homogènes. Sa popularité est dûe au fait que l'algorithme bien connu des k-moyennes optimise ce critère et trouve un optimum local. Trouver un optimum global pour ce critère est un problème NP-Difficile et trouver une bonne borne inférieure est aussi difficile [1]. La meilleure approche exacte pour ce critère est basée sur la programmation linéaire sur des entiers et la génération de colonnes [2]. Elle a été récemment généralisée pour intégrer des contraintes utilisateur [3]. Dans cet ar-

ticle, nous proposons un calcul de borne inférieure et développons un algorithme de filtrage pour une nouvelle contrainte *wcss* qui caractérise ce critère. Des expérimentations sur des bases de données classiques montrent que notre approche basée sur la programmation par contrainte a une meilleure performance que l'approche fondée sur la programmation linéaire sur des entiers et la génération de colonnes. De plus, ce critère s'intègre dans un cadre général, qui permet à l'utilisateur de construire différentes tâches de clustering en combinant différents types de contraintes utilisateur.

Dans la suite du papier, nous présentons, en section 2, des notions préliminaires sur le clustering sous contraintes utilisateur et des travaux connexes. Nous détaillons le modèle en programmation par contraintes dans la section 3 et l'algorithme de filtrage pour la nouvelle contrainte *wcss* dans la section 4. Les expérimentations sont présentées dans la section 5 ; la section 6 contient une discussion sur des travaux futurs et la conclusion.

2 Préliminaires

2.1 Clustering sous contraintes utilisateur

Le clustering vise à regrouper les données en un ensemble de classes ou clusters, de façon à ce que les objets d'une même classe aient une forte similarité entre eux et diffèrent fortement des objets des autres classes. Le clustering est souvent défini par un problème d'optimisation, i.e. trouver une partition des objets qui optimise un critère donné. Considérons une base de données de n objets $\mathcal{O} = \{o_1, \ldots, o_n\}$ et une mesure de dissimilarité $d(o, o')$ entre deux objets $o, o' \in \mathcal{O}$. Une partition Δ des objets en k classes $C_1, \ldots, C_k$ est telle que : (1) pour tout $c \in [1, k]$ [1], $C_c \neq \emptyset$, (2) $\cup_c C_c = \mathcal{O}$ et (3) pour tout $c \neq c'$, $C_c \cap C_{c'} = \emptyset$. Le critère à optimiser peut être entre autres :

- minimiser le diamètre maximal des clusters, où le diamètre maximal d'une partition Δ est défini par
$$D(\Delta) = \max_{c \in [1,k]} \max_{o,o' \in C_c} d(o, o')$$
- maximiser la marge minimale entre clusters, où la marge minimale de Δ est définie par
$$S(\Delta) = \min_{c,c' \in [1,k]} \min_{o \in C_c, o' \in C_{c'}} d(o, o')$$
- minimiser la somme des dissimilarités intra-cluster WCSD (Within-Cluster Sum of Dissimilarities)
$$WCSD(\Delta) = \sum_{c \in [1,k]} \frac{1}{2} \sum_{o,o' \in C_c} d(o, o')$$
- minimiser la somme des carrés intra-cluster WCSS (Within-Cluster Sum of Squares)
$$WCSS(\Delta) = \sum_{c \in [1,k]} \sum_{o \in C_c} ||o - m_c||^2$$
où pour chaque $c \in [1, k]$, m_c est le centre du cluster C_c. Dans l'espace euclidien, lorsque la dissimilarité est définie par $d(o, o') = ||o - o'||^2$, nous avons la relation suivante :
$$WCSS(\Delta) = \sum_{c \in [1,k]} \frac{\frac{1}{2} \sum_{o,o' \in C_c} d(o, o')}{|C_c|}$$

La classification non supervisée est un problème NP-Difficile pour tous ces critères sauf pour la maximisation de la marge minimale. La plupart des algorithmes classiques utilisent des heuristiques et trouvent un optimum local [12]. Par exemple l'algorithme bien connu des k-moyennes (*k-means*) trouve un optimum local pour le critère WCSS. Cependant plusieurs optima peuvent exister et certains pouvant être plus proches de la solution recherchée. Afin de mieux modéliser la tâche d'apprentissage, mais aussi en espérant réduire la complexité, des contraintes définies par l'utilisateur sont ajoutées. On parle alors de la classification non supervisée sous contraintes utilisateur. Ces contraintes peuvent porter sur les classes (contraintes de classe) ou sur les objets (contraintes objet). Deux types de contraintes objets, introduites dans [16], sont couramment utilisés : must-link ou cannot-link. Une contrainte must-link sur deux objets spécifie qu'ils doivent être dans la même classe et une contrainte cannot-link spécifie qu'ils ne doivent pas être dans la même classe.

Les contraintes de classe imposent des restrictions sur les classes. La contrainte de capacité minimale (resp. maximale) exige que chaque classe ait au moins (resp. au plus) un nombre donné α (resp. β) d'objets : $\forall c \in [1, k],\ |C_c| \geq \alpha$ (resp. $\forall c \in [1, k],\ |C_c| \leq \beta$). La contrainte de diamètre maximal spécifie une borne supérieure γ sur le diamètre de chaque cluster : $\forall c \in [1, k], \forall o, o' \in C_c,\ d(o, o') \leq \gamma$. La contrainte de marge minimale, aussi appelée δ-contrainte dans [9], spécifie une borne inférieure δ sur la marge entre deux classes : $\forall c \in [1, k],\ \forall c' \neq c, \forall o \in C_c, o' \in C_{c'}, d(o, o') \geq \delta$. Une contrainte de densité, extension de la ϵ-contrainte introduite dans [9], demande que chaque point o ait dans son voisinage de rayon ϵ au moins m points de la même classe : $\forall c \in [1, k], \forall o \in C_c, \exists o_1, .., o_m \in C_c, o_j \neq o \wedge d(o, o_j) \leq \epsilon$.

2.2 Travaux connexes

Le clustering avec le critère de minimisation de WCSS est un problème NP-Difficile [1]. L'algorithme

1. Nous utilisons $[1, k]$ pour désigner l'ensemble $\{1, .., k\}$.

populaire des k-moyennes trouve un optimum local pour ce critère. Cet algorithme commence avec une partition aléatoire et répète deux étapes : (1) associer chaque objet au cluster dont le centre est le plus proche, (2) calculer les nouveaux centres des clusters. Ces deux étapes sont répétées jusqu'à ce qu'il n'y ait plus de changement d'affectation de points. Le résultat trouvé dépend donc de la partition aléatoire initiale. Quant aux méthodes exactes pour résoudre ce problème, nous pouvons citer l'algorithme de recherche par *branch-and-bound* répétitif RBBA (Repetitive Branch and Bound Algorithm) [5] et l'algorithme basé sur la programmation linéaire sur des entiers et la génération de colonnes [2]. Actuellement les approches exactes les plus efficaces peuvent résoudre ce problème pour des bases d'environ 2300 objets [2].

Lorsque des contraintes utilisateur s'ajoutent, l'algorithme COP-kmeans étend l'algorithme k-means pour traiter des contraintes must-link ou cannot-link [17]. Cet algorithme modifie l'algorithme des k-moyennes en affectant chaque objet au cluster dont le centre est le plus proche, parmi les clusters satisfaisant les contraintes. S'il n'existe pas de cluster satisfaisant les contraintes, l'algorithme s'arrête. Cet algorithme est glouton et très rapide, mais il peut ne pas trouver de solution satisfaisant toutes les contraintes même lorsqu'une telle solution existe [10].

Récemment, un cadre général pour minimiser ce critère WCSS et intégrant différents types de contraintes utilisateur a été développé [3]. Il étend l'approche [2] fondé sur la programmation linéaire sur des entiers et la génération de colonnes. Ce cadre permet de trouver une solution exacte, qui est un optimum global satisfaisant les contraintes utilisateur. Cependant ce cadre ne permet de traiter que le critère WCSS.

Dans des travaux récents, nous avons développé un cadre général pour le clustering sous contraintes utilisateur à base de la programmation par contraintes [6, 7]. Ce cadre permet de traiter différents critères d'optimisation : minimiser le diamètre maximal, maximiser la marge minimale ou minimiser WCSD, tout en intégrant tous les types de contraintes utilisateur populaires. En plus de traiter des problèmes de clustering mono-critère, la généricité du cadre permet de l'utiliser dans des problèmes de clustering bi-critère, comme minimiser le diamètre et maximiser la marge sous contraintes utilisateur [8]. Dans ce papier nous développons ce cadre pour traiter le clustering avec la minimisation de la somme des carrés WCSS sous contraintes utilisateur.

3 Modélisation des tâches de clustering sous contraintes utilisateur

Nous présentons un modèle en programmation par contraintes pour le clustering, intégrant le critère WCSS et des contraintes utilisateur. Ce modèle étend le cadre que nous avons développé pour les critères de diamètre, de la marge ou de WCSD. Pour plus de détails sur ces critères, nous renvoyons le lecteur vers [6, 7, 8]. Le modèle permet de formaliser les tâches de clustering avec ces critères pour un nombre de clusters k variant entre k_{min} et k_{max} clusters ($k_{min} \leq k \leq k_{max}$). Les valeurs k_{min} et k_{max} sont fixées par l'utilisateur. Si l'utilisateur souhaite k clusters, il suffit de fixer $k_{min} = k_{max} = k$.

Dans ce modèle, les clusters sont identifiés par leur indices, qui varient de 1 à k pour une partition en k clusters. Sans perdre de généralité, pour désigner le point o_i nous utilisons son indice i. Pour représenter l'affectation des points aux clusters, nous utilisons des variables entières $G_1, \ldots, G_n$, avec pour domaine $Dom(G_i)$ l'ensemble des entiers $[1, k_{max}]$. Une affectation $G_i = c$ signifie que le point i est affecté au cluster numéro c. Soit $\mathcal{G}$ le tableau des variables $[G_1, \ldots, G_n]$. Pour représenter le critère de la somme des carrés, nous introduisons une variable V ayant pour domaine l'intervalle $[0, \infty)$.

Les contraintes de la PPC permettent d'exprimer les contraintes de partition et les contraintes utilisateur.

3.1 Contraintes de partition

Toute affectation complète des variables de $\mathcal{G}$ définit une partition de points en clusters. Cependant, une partition peut se représenter par plusieurs affectations différentes. Par exemple, à partir d'une affectation des variables de $\mathcal{G}$, si l'on fait un échange où toutes les variables G_i qui ont la valeur c_1 prennent la valeur c_2 et les variables G_j qui ont la valeur c_2 prennent la valeur c_1, nous obtenons une nouvelle affectation mais qui représente toujours la même partition, du point de vue des objets composant les classes. Un autre cas où l'on obtient la même partition avec une affectation différente des variables de $\mathcal{G}$ est lorsque les variables ayant une valeur c_1 prennent une valeur c_3 qui n'est pas utilisée dans l'affectation. De telles situations représentent des symétries de valeurs. Pour casser ces symétries, nous imposons que les clusters formés par les variables G_i soient tels que le numéro 1 est l'indice du premier cluster créé, et un numéro c, avec $c > 1$ est utilisé pour un nouveau cluster si et seulement si le numéro $c - 1$ est déjà utilisé. Une méthode directe pour représenter ces conditions [7] est de poser la contrainte $G_1 = 1$ et les contraintes $G_i \leq \max_{j \in [1, i-1]}(G_j) + 1$, pour $i \in [2, n]$. Cependant, afin d'obtenir de meilleures

interactions et propagations, il est intéressant de présenter ces relations par une contrainte globale avec un bon algorithme de filtrage. La contrainte *precede* [13] permet de réaliser cela :

$$precede(\mathcal{G}, [1, \ldots, k_{max}]).$$

Cette contrainte impose que $G_1 = 1$ et de plus, si $G_i = c$ avec $1 < c \leq k_{max}$, alors il doit exister un indice $j < i$ tel que $G_j = c - 1$.

Le fait qu'il y ait au moins k_{min} clusters signifie que les valeurs de 1 à k_{min} doivent apparaître dans G_i. Avec l'utilisation de la contrainte *precede*, il suffit qu'au moins une variable G_i soit égale à k_{min}. Ceci est représenté par $\#\{i \mid G_i = k_{min}\} \geq 1$, ou par la contrainte

$$atleast(1, \mathcal{G}, k_{min}).$$

Puisque le domaine des variables G_i est $[1, k_{max}]$, il existe au plus k_{max} clusters. Si l'on veut exactement k clusters, il suffit de mettre $k_{min} = k_{max} = k$.

3.2 Contraintes utilisateur

Toutes les contraintes utilisateur populaires peuvent être intégrées dans le modèle.

- Une contrainte must-link sur deux points i, j est représentée par : $G_i = G_j$.
- Une contrainte cannot-link sur i, j est représentée par : $G_i \neq G_j$.
- Taille minimale des clusters α : chaque point doit être dans un cluster avec au moins α points (y compris ce point). Pour chaque $i \in [1, n]$, la valeur de la variable G_i doit donc apparaître au moins α fois dans $\mathcal{G}$, *i.e.* $\#\{j \mid G_j = G_i\} \geq \alpha$. Pour chaque $i \in [1, n]$, on pose la contrainte : $atleast(\alpha, \mathcal{G}, G_i)$. Cette contrainte permet aussi de mettre une borne sur le nombre de clusters. En effet, on peut poser $G_i \leq \lfloor n/\alpha \rfloor$, pour chaque $i \in [1, n]$.
- Taille maximale β des clusters : chaque nombre $c \in [1, k_{max}]$ doit apparaître dans $\mathcal{G}$ au plus β fois (c'est aussi vrai pour les numéros non utilisés $c \in [k_{min} + 1, k_{max}]$, qui apparaissent 0 fois), *i.e.* $\#\{i \mid G_i = c\} \leq \beta$. Donc, pour chaque $c \in [1, k_{max}]$, nous posons : $atmost(\beta, \mathcal{G}, c)$. Cette contrainte implique aussi : $G_i \geq \lceil n/\beta \rceil$, pour tout $i \in [1, n]$.
- Une contrainte de marge impose que la marge entre deux clusters doit être au moins δ. Donc pour chaque couple $i < j \in [1, n]$ tel que $d(i, j) < \delta$, la contrainte $G_i = G_j$ est posée.
- Une contrainte de diamètre impose que le diamètre de chaque cluster ne doit pas dépasser γ. Pour chaque couple $i < j \in [1, n]$ tel que $d(i, j) > \gamma$, on pose : $G_i \neq G_j$.
- Une contrainte de densité exprime que chaque point doit avoir dans son voisinage de rayon ϵ au moins m points dans le même cluster que lui. Donc pour chaque $i \in [1, n]$, l'ensemble de variables dans le voisinage de rayon ϵ est calculé $\mathcal{N}_{i\epsilon} = \{G_j \mid d(i, j) \leq \epsilon\}$ et cette contrainte est posée par : $atleast(m, \mathcal{N}_{i\epsilon}, G_i)$.

3.3 Critère d'optimisation

Pour représenter le fait que V est la somme des carrés des clusters formés par les variables de $\mathcal{G}$, nous développons une contrainte globale $wcss(\mathcal{G}, V, d)$. L'algorithme de filtrage de cette contrainte est présenté dans la section 4. La valeur de V est à minimiser.

3.4 Stratégie de recherche

Le branchement est effectué sur les variables de $\mathcal{G}$. Une stratégie mixte est utilisée. Pour trouver la première solution, une recherche gloutonne est utilisée : à chaque branchement, une variable G_i et une valeur $c \in Dom(G_i)$ telles que l'affectation $G_i = c$ augmente le moins possible V sont choisies. Lorsque la première solution est trouvée, la valeur de V dans cette solution constitue une borne supérieure de la variable V. Après avoir trouvé la première solution, la stratégie de recherche change ensuite en une autre stratégie. Dans cette stratégie, à chaque branchement, pour chaque variable G_i qui n'est pas instanciée, pour chaque valeur $c \in Dom(G_i)$ nous calculons la valeur a_{ic}, qui est la valeur ajoutée à V dans le cas où le point i est affecté au cluster c. Pour chaque variable G_i non instanciée, soit $a_i = \min_{c \in Dom(G_i)} a_{ic}$. La valeur a_i représente donc la valeur minimale ajoutée à V lorsque le point i est affecté à un cluster. Puisque chaque point doit être affecté à un cluster, à chaque branchement, la variable G_i choisie sera celle dont la valeur a_i est maximale, et pour cette variable, la valeur c choisie sera celle dont a_{ic} est minimale. Deux branches sont ensuite créées, une avec $G_i = c$ et une avec $G_i \neq c$. Avec cette stratégie, on tente de détecter rapidement l'échec et dans le cas sans échec, à obtenir une solution avec la valeur V la plus petite possible.

4 Algorithme de filtrage

Nous développons un algorithme de filtrage pour une nouvelle contrainte $wcss(\mathcal{G}, V, d)$. Cette contrainte assure que V représente la somme des carrés des clusters formés par l'instanciation des variables de $\mathcal{G}$. Nous présentons dans cette section le calcul de la borne inférieure de V et l'algorithme pour filtrer les domaines de V et des variables de $\mathcal{G}$, lorsqu'on a une instanciation partielle des variables de $\mathcal{G}$. Puisque la variable

V représente la fonction objectif à minimiser, cette contrainte est de type contrainte globale d'optimisation [11, 15]. L'algorithme de filtrage permet de filtrer non seulement le domaine de la variable objectif V, mais aussi le domaine des variables décisionnelles G_i.

Une instanciation partielle de variables de $\mathcal{G}$ correspond au cas où certain nombre de points sont déjà affectés aux clusters et il reste encore des points non affectés. Soit U l'ensemble des points non affectés et soit $q = |U|$. Soit $\mathcal{C}$ l'ensemble de tous les clusters et soit $k = \max_i\{c \mid c \in Dom(G_i)\}$. L'entier k représente le plus grand numéro de clusters restant dans les domaines de variables de $\mathcal{G}$ et donc le nombre maximum de clusters possible. Pour chaque cluster C_c ($c \in [1, k]$), soit n_c le nombre de points déjà affectés au cluster C_c ($n_c = |C_c|$) et soit $S_1(C_c)$ la somme des dissimilarités des points qui sont déjà affectés au cluster C_c : $S_1(C_c) = \frac{1}{2}\sum_{i,j\in C_c} d(i, j)$.

4.1 Calcul de la borne inférieure de V

L'objectif est de calculer une borne inférieure de la somme des carrés V lorsque l'on affecte tout l'ensemble des points non affectés U dans les clusters $C_1, \ldots, C_k$. Ce calcul se fait en deux étapes :

1. Pour chaque $m \in [0, q]$ et pour chaque $c \in [1, k]$, nous calculons une borne inférieure de la somme des carrés $\underline{V}(C_c, m)$ pour le cas où l'on ajoute m points quelconques de U dans le cluster C_c.
2. Pour chaque $m \in [0, q]$ et pour chaque $c \in [2, k]$, nous calculons une borne inférieure de la somme des carrés $\underline{V}(C_1 \ldots C_c, m)$ pour le cas où l'on ajoute m points quelconques de U dans c clusters $C_1, \ldots, C_c$.

Le calcul de la borne inférieure de V s'effectue par l'algorithme 1. Nous détaillons ci-dessous les deux étapes de calcul.

Ajout de m points de U à un cluster C_c. Si l'on choisit un ensemble $U' \subseteq U$ avec $|U'| = m$ et si l'on ajoute les points de U' dans le cluster C_c, la somme des carrés du cluster sera

$$V(C_c, U') = \frac{S_1(C_c) + S_2(C_c, U')}{n_c + m}$$

Ici $S_1(C_c)$ est la somme des dissimilarités entre les points déjà dans C_c et $S_2(C_c, U')$ est la somme des dissimilarités liées aux points de U'. La valeur de $S_1(C_c)$ est connue. Si l'on connait l'ensemble U' la valeur de $S_2(C_c, U')$ sera aussi connue. Elle est calculée par :

$$S_2(C_c, U') = \sum_{u\in U', v\in C_c} d(u, v) + \frac{1}{2}\sum_{u,v\in U'} d(u, v)$$

Algorithme 1 : Borne_inf()

input : une instanciation partielle de variables de $\mathcal{G}$, un ensemble $U = \{i \mid G_i$ non instanciée$\}$
output : une borne inférieure pour V

1 **foreach** $x \in U$ **do**
2 trier $u \in U$ dans l'ordre croissant de $d(x, u)$
3 **foreach** $c \in [1, k]$ **do**
4 **if** $c \notin Dom(G_x)$ **then**
5 $s_2[x, c] \leftarrow \infty$
6 **else**
7 $s_2[x, c] \leftarrow 0$
8 **foreach** $v \in [1, n]$ **do**
9 **if** G_v *est instanciée et* $G_v \in Dom(G_x)$ **then**
 $s_2[x, val(G_v)] = s_2[x, val(G_v)] + d(x, v)$
10 $s_3[x, 0] \leftarrow 0$;
11 **for** $m \leftarrow 1$ **to** q **do**
12 $s_3[x, m] \leftarrow s_3[x, m-1] + d(x, u_i)/2$
13 **foreach** $c \in [1, k]$ **do**
14 **foreach** $m \in [0, q]$ **do**
15 **foreach** $x \in U$ **do**
16 $s[x] = s_2(x, c) + s_3(x, m)$
17 trier tableau s dans l'ordre croissant
18 $\underline{S_2}(C_c, m) \leftarrow \sum_{i=1}^{m} s[i]$
19 **if** $n_c + m = 0$ **then**
20 $\underline{V}(C_c, m) \leftarrow 0$
21 **else**
22 $\underline{V}(C_c, m) \leftarrow$ $(S_1(C_c) + \underline{S_2}(C_c, m))/(n_c + m)$
23 **for** $c \leftarrow 2$ **to** k **do**
24 **for** $m \leftarrow 1$ **to** q **do**
25 $\underline{V}(C_1 \ldots C_c, m) \leftarrow$ $\min_{i\in[0,m]}(\underline{V}(C_1 \ldots C_{c-1}, i) + \underline{V}(C_c, m-i))$
26 retourner $\underline{V}(C_1 \ldots C_k, q)$

Mais $S_2(C_c, U')$ devient inconnu lorsque l'ensemble U' n'est pas fixé. Pour tous les sous-ensembles U' de taille m, on peut cependant calculer une borne inférieure $\underline{S_2}(C_c, m)$ comme suit.

Chaque point $x \in U$, dans le cas où il est affecté au cluster C_c avec $m-1$ autres points, va contribuer une somme $s(x, c, m) = s_2(x, c) + s_3(x, m)$, où

- $s_2(x, c)$ représente la somme des dissimilarités entre x et les points de C_c. Si $c \notin Dom(G_x)$ alors $s_2(x, c) = \infty$, car x ne peut pas être affecté au cluster C_c. Sinon :

$$s_2(x, c) = \sum_{v \in C_c} d(x, v)$$

  Cette valeur $s_2(x, c)$ vaut 0 si le cluster C_c est vide. Lignes 3-9 de l'algorithme 1 calculent $s_2(x, c)$.
- $s_3(x, m)$ représente la moitié de la somme des dissimilarités $d(x, z)$, pour tous les $m-1$ autres points z ; si l'on trie tous les points $u \in U$ dans l'ordre croissant des dissimilarités $d(x, u)$, et l'on désigne par u_i le i-eme point dans cet ordre, on a

$$s_3(x, m) \;\geq\; \frac{1}{2} \sum_{i=1}^{m-1} d(x, u_i)$$

  cf. lignes 11-12 de l'algorithme 1.

Pour chaque $c \in [1, k]$ et $m \in [0, q]$, la valeur $s(x, c, m)$ est représentée par $s[x]$ (ligne 15). La borne inférieure $\underline{S_2}(C_c, m)$ est donc la somme des m plus petites contributions $s(x, c, m)$, pour tous les points $x \in U$. La borne inférieure $\underline{V}(C_c, m)$ est calculée par :

$$\underline{V}(C_c, m) \;=\; \frac{S_1(C_c) + \underline{S_2}(C_c, m)}{n_c + m} \qquad (1)$$

Ce calcul est réalisé pour tout $c \in [1, k]$ et pour tout $m \in [0, q]$ (lignes 13 à 21 de l'algorithme 1).

Ajout de m points de U aux c clusters $C_1, .., C_c$. Pour $m \in [0, q]$ et $c \in [1, k]$, calculons une borne inférieure $\underline{V}(C_1..C_c, m)$ de tous les cas possibles où l'on ajoute m points de U aux clusters $C_1, ..., C_c$. Si l'on choisit m points $x_1, \ldots, x_m \in U$ et si l'on ajoute i points $x_1, \ldots, x_i$ aux $c-1$ clusters $C_1, \ldots, C_{c-1}$ et les autres $m-i$ points au cluster C_c, la somme des carrés $V(C_1..C_c, \{x_1, .., x_m\})$ sera égale à :

$$V(C_1..C_{c-1}, \{x_1, ..x_i\}) + V(C_c, \{x_{i+1}, .., x_m\})$$

Lorsque les points $x_1, \ldots, x_m$ sont des points quelconques de U, la valeur exacte n'est pas connue. Cependant :

$$V(C_1..C_c, m) \geq \min_{i \in [0,m]} (V(C_1..C_{c-1}, i) + V(C_c, m-i))$$

La borne inférieure $\underline{V}(C_1..C_c, m)$ est donc calculée par :

$$\underline{V}(C_1..C_c, m) = \min_{i \in [0,m]} (\underline{V}(C_1..C_{c-1}, i) + \underline{V}(C_c, m-i)) \qquad (2)$$

Ce calcul est réalisé par un programme dynamique pour tout $c \in [2, k]$ et pour tout $m \in [0, q]$ (lignes 23 à 25 de l'algorithme 1). Soit $[V.lb, V.ub)$ l'intervalle représentant le domaine de la variable V. La borne inférieure $V.lb$ est donc fixée à $\underline{V}(C_1 \ldots C_k, q)$, la borne inférieure lorsque l'on affecte l'ensemble de points U aux clusters $C_1, \ldots, C_k$. La complexité de l'algorithme est $O(kqn \log q)$.

Notons qu'avec la formule (2), pour tout $c \in [1, k]$, pour tout $m \in [0, q]$, $\underline{V}(C_1..C_c, m)$ est égal à

$$\min_{m_1 + .. + m_c = m} (\underline{V}(C_1, m_1) + \cdots + \underline{V}(C_c, m_c)) \qquad (3)$$

4.2 Filtrage

Etant donnée une instanciation partielle des variables de $\mathcal{G}$, la borne inférieure $V.lb$ de la variable V est calculée. Nous présentons le filtrage du domaine des variables G_i qui ne sont pas instanciées dans l'algorithme 2. Cet algorithme permet donc de filtrer le domaine de V qui représente la fonction objectif, mais aussi le domaine des variables décisionnelles G_i.

Dans cet algorithme, pour chaque valeur $c \in [1, k]$, pour chaque variable G_i qui n'est pas instanciée, si $c \in Dom(G_i)$, sous l'hypothèse que le point i soit affecté au cluster C_c, on calcule une nouvelle borne inférieure de V. Soit C'_c le cluster $C_c \cup \{i\}$ et soit $\mathcal{C}' = \{C_l \mid l \neq c\} \cup \{C'_c\}$. La nouvelle borne inférieure V' de V sera la valeur de $\underline{V}(\mathcal{C}', q-1)$, car il reste $q-1$ points dans l'ensemble U à affecter aux k clusters. Suivant le même principe que (2), on a :

$$\underline{V}(\mathcal{C}', q-1) = \min_{m \in [0, q-1]} (\underline{V}(\mathcal{C}' \backslash \{C'_c\}, m) + \underline{V}(C'_c, q-1-m))$$

Il faudra donc réviser les bornes inférieures $\underline{V}(\mathcal{C}' \backslash \{C'_c\}, m)$ et $\underline{V}(C'_c, m)$, pour tout $m \in [0, q-1]$. Ces calculs sont présentés dans le reste de cette sous-section. La nouvelle borne inférieure V' est calculée par la ligne 8 de l'algorithme 2. Ensuite, puisque $Dom(V) = [V.lb, V.ub)$, si $V' \geq V.ub$, la valeur c est inconsistante et sera supprimée du $Dom(G_i)$ (lignes 9-10). La complexité de l'algorithme 2 est aussi de $O(kqn \log q)$.

Calcul de $\underline{V}(C'_c, m)$. Rappelons que C'_c est le cluster C_c auquel on a ajouté le point i, et $\underline{V}(C'_c, m)$ est la borne inférieure de la somme des carrés de C'_c lorsqu'on ajoute m points quelconques de U dans C'_c. D'après (1) :

$$\underline{V}(C'_c, m) = \frac{S_1(C'_c) + \underline{S_2}(C'_c, m)}{n_c + 1 + m}$$

La valeur de $S_1(C'_c)$ est

$$S_1(C'_c) = S_1(C_c) + s_2(i,c)$$

La valeur de $\underline{S_2}(C'_c, m)$ peut être révisée à partir de $\underline{S_2}(C_c, m)$ par :

$$\underline{S_2}(C'_c, m) = \underline{S_2}(C_c, m) + s_3(i,m)$$

Par (1), on a donc (ligne 7 de l'algorithme 2) :

$$\underline{V}(C'_c, m) = \frac{(n_c+m)\underline{V}(C_c,m) + s_2(i,c) + s_3(i,m)}{n_c+m+1}$$

Calcul de $\underline{V}(\mathcal{C}'\backslash\{C'_c\}, m)$. Cette valeur représente la borne inférieure lorsque l'on ajoute m points de $U\backslash\{i\}$ dans les clusters différents de C'_c. D'après (3), pour tout $q' \in [m, q]$ on a :

$$\underline{V}(\mathcal{C}, q') = \min_{m+m'=q'} (\underline{V}(\mathcal{C}\backslash\{C_c\}, m) + \underline{V}(C_c, m'))$$

donc pour tout $q' \in [m, q]$ et avec $m + m' = q'$, on a :

$$\underline{V}(\mathcal{C}, m+m') \leq \underline{V}(\mathcal{C}\backslash\{C_c\}, m) + \underline{V}(C_c, m')$$

ce qui correspond à :

$$\underline{V}(\mathcal{C}\backslash\{C_c\}, m) \geq \underline{V}(\mathcal{C}, m+m') - \underline{V}(C_c, m')$$

Nous avons donc :

$$\underline{V}(\mathcal{C}\backslash\{C_c\}, m) \geq \max_{m' \in [0, q-m]} (\underline{V}(\mathcal{C}, m+m') - \underline{V}(C_c, m'))$$

Nous avons aussi :

$$\underline{V}(\mathcal{C}'\backslash\{C'_c\}, m) \geq \underline{V}(\mathcal{C}\backslash\{C_c\}, m)$$

car $\mathcal{C}'\backslash\{C'_c\}$ et $\mathcal{C}\backslash\{C_c\}$ désigne le même ensemble de clusters, $\underline{V}(\mathcal{C}'\backslash\{C'_c\}, m)$ est calculé pour m points quelconques de $U\backslash\{i\}$, pendant que $\underline{V}(\mathcal{C}\backslash\{C_c\}, m)$ est calculé pour m points quelconques de U. On peut donc fixer une nouvelle borne inférieure (ligne 4 de l'algorithme 2) :

$$\underline{V}(\mathcal{C}'\backslash\{C'_c\}, m) = \max_{m' \in [0, q-m]} (\underline{V}(\mathcal{C}, m+m') - \underline{V}(C_c, m'))$$

Algorithme 2 : Filtrage pour $wcss(\mathcal{G}, V, d)$

```
1  V.lb ← Borne_inf()
2  foreach c ∈ [1, k] do
3      foreach m ∈ [0, q − 1] do
4          V(C'\{C'_c}, m) ←
           max_{m'∈[0,q−m]}(V(C, m + m') − V(C_c, m'))
5      foreach i ∈ [1, n] tel que |Dom(G_i)| > 1 et
       c ∈ Dom(G_i) do
6          for m ← 0 to q − 1 do
7              V(C'_c, m) ← ((n_c + m)V(C_c, m) +
               s_2(i, c) + s_3(i, m))/(n_c + m + 1)
8          V' ← min_{m∈[0,q−1]}(V(C'\{C'_c}, m) +
           V(C'_c, q − 1 − m))
9          if V' ≥ V.ub then
10             supprimer c du Dom(G_i)
```

5 Expérimentations

Notre modèle est implanté en utilisant la librairie Gecode (`http://www.gecode.org`) version 4.2.7. Cette librairie supporte à la fois des variables flottantes et entières. Les expérimentations sont réalisées sur un ordinateur 3.0 GHz Core i7 Intel avec 8 Go de mémoire sous Ubuntu. Tous nos programmes sont disponibles sur `http://cp4clustering.com`. Nous considérons les bases Iris et Wine du dépôt de données UCI (`http://archive.ics.uci.edu/ml`). La base Iris est composée de 150 objets et 3 classes, la base Wine de 178 objets et 3 classes. Nous comparons notre modèle avec l'approche basée sur la programmation linéaire sur des entiers et la génération de colonnes [3] pour le clustering avec le critère WCSS et avec les contraintes utilisateur. Nous comparons également notre modèle avec l'algorithme RBBA [4] dans le cas sans contraintes utilisateur et avec l'algorithme COP-kmeans [17] qui trouve des solutions approchées.

5.1 WCSS avec contraintes utilisateur

Pour générer les contraintes utilisateur must-link et cannot-link, des paires d'objets sont retirées aléatoirement et les classes réelles des objets sont considérées. Si les objets sont de la même classe, une contrainte must-link est générée, sinon une contrainte cannot-link est générée. Les contraintes must-link ou cannot-link sont générées jusqu'à ce que le nombre de contraintes souhaité soit atteint. Pour chaque nombre de contraintes, nous générons 5 ensembles différents pour effectuer le test. Nous considérons pour chaque test la valeur de WCSS, le temps total d'exécution et l'indice Rand de la solution trouvée. Puisque le test est effectué 5 fois pour chaque nombre de contraintes, nous reportons la valeur moyenne et l'écart-type des tests. Une limite de temps est fixée à 30 minutes. Le signe "-" dans les tableaux indique que le temps limite est dépassé sans qu'une solution optimale soit trouvée et prouvée.

Le tableau 1 donne, pour la base Iris, la moyenne sur les 5 tests du temps d'exécution total en secondes ainsi que de la valeur de l'indice de Rand [14], ainsi que le pourcentage de l'écart type par rapport à la moyenne

#c	T moy	% ecart	RI moy	% ecart
0	888,99	0,83	0,879	0
50	332,06	78,96	0,940	1,66
100	7,09	40,18	0,978	1,68
150	0,31	36,39	0,989	0,66
200	0,07	24,83	0,992	0,66
250	0,05	10,63	0,996	0,70
300	0,04	9,01	0,998	0,35

TABLE 1 – Base Iris avec #c contraintes must-link

#c	CP	% ecart	GC	% ecart
0	888,99	0,83	-	-
50	332,06	78,96	-	-
100	7,09	40,18	62	74,24
150	0,31	36,39	0,45	44,55
200	0,06	24,83	0,11	48,03
250	0,05	10,63	0,06	35,56
300	0,04	9,01	0,04	19,04

TABLE 2 – Base Iris avec #c must-link

$(100\sigma/\mu)$. L'indice de Rand mesure la similarité entre une partition P et la partition réelle P^* de la base de données. Il est défini par :

$$Rand = \frac{a+b}{a+b+c+d}$$

où a et b sont les nombres de paires de points pour lesquelles P et P^* s'accordent (a nombre de paires qui se retrouvent dans la même classe dans P et dans P^*, b nombre de paires dans différentes classes), c et d sont les nombres de paires de points pour lesquelles P et P^* ne s'accordent pas (même classe dans P mais différentes classes dans P^* et vice versa). L'indice Rand varie entre 0 et 1 et plus les deux partitions s'accordent, plus il est proche de 1.

On peut remarquer que pour cette base, les contraintes utilisateur ajoutées permettent de réduire le temps d'exécution et de trouver des solutions de très bonne qualité.

Pour la base Iris, notre modèle (CP) peut trouver une solution optimale pour le problème de minimisation de WCSS sans contraintes utilisateur. L'approche par la programmation linéaire sur des entiers et la génération de colonnes (GC) [3] ne peut pas prouver l'optimalité pour ce problème avec moins de 100 contraintes must-link. Les tableaux 2 et 3 reportent les temps d'exécution en moyenne en secondes des tests des deux approches et le pourcentage de l'écart type des tests par rapport au temps moyen. Le tableau 2 présente les cas où seules des contraintes must-link sont ajoutées et le tableau 3 les cas où l'on ajoute le même nombre $\#c$ de contraintes must-link (ML) et cannot-link (CL). On peut constater que notre modèle est le plus efficace dans ces cas.

Pour la base Wine, les deux approches ne peuvent pas prouver l'optimalité en moins de 30 minutes avec moins de 150 contraintes must-link, comme le montre le tableau 4 avec les temps moyen en secondes. On peut constater que pour la base Wine, notre modèle est le plus efficace lorsque des contraintes must-link sont ajoutées.

#c	CP	% ecart	GC	% ecart
25	969,33	51,98	-	-
50	43,85	46,67	-	-
75	4,97	150	-	-
100	0,41	49,8	107	72,35
125	0,09	52,07	4,4	95,85
150	0,06	22,6	0,8	50

TABLE 3 – Base Iris avec #c ML et #c CL

Notre approche à base de programmation par contraintes permet aussi de mieux exploiter les contraintes cannot-link, comme le montre les résultats des tableaux 5 et 6. Ces tableaux présentent le temps en moyen en secondes et le pourcentage des cas résolus parmi tous les tests. Le temps de calcul varie beaucoup en fonction des contraintes ajoutées. Par exemple pour la base Iris, le tableau 5 montre que notre modèle est capable de trouver et de prouver l'optimalité pour environ 60% des cas, alors que GC ne peut résoudre aucun cas. Pour la base Wine, le tableau 6 montre qu'avec 100 contraintes must-link et 100 contraintes cannot-link, notre modèle peut résoudre tous les cas alors que GC ne peut résoudre aucun cas. Lorsque 125 contraintes must-link et 125 contraintes cannot-link sont ajoutées, les deux approches peuvent résoudre tous les cas, mais notre approche prend moins de temps.

#c	CP	GC
150	6,84	12,98
200	0,11	0,32
250	0,08	0,11
300	0,08	0,06

TABLE 4 – Base Wine avec #c must-link

#c	CP	%res CP	GC	%res GC
50	1146,86	20%	-	0%
100	719,53	80%	-	0%
150	404,77	60%	-	0%
200	1130,33	40%	-	0%
250	172,81	60%	-	0%
300	743,64	60%	-	0%

TABLE 5 – Base Iris avec #c contraintes cannot-link

#c	CP	%res CP	GC	% res GC
100	10,32	100%	-	0%
125	0,35	100%	497,6	100%
150	0,12	100%	13,98	100%

TABLE 6 – Base Wine avec #c ML et #c CL

5.2 Comparaisons avec COP-kmeans et RBBA

L'algorithme RBBA basé sur la recherche *branch-and-bound* répétitive [4] permet de trouver une solution exacte qui optimise globalement le critère WCSS. Cet algorithme prend 0.53s pour trouver et prouver la solution optimale pour la base Iris et 35.56s pour la base Wine. Cependant, il ne permet pas d'intégrer des contraintes utilisateur.

Lorsque les contraintes must-link et cannot-link s'ajoutent, l'algorithme COP-kmeans [17] basé sur une recherche gloutonne cherche une solution approchée pour le critère WCSS satisfaisant toutes les contraintes. Dans le cas où il n'y a que des contraintes must-link, COP-kmeans trouve toujours une partition satisfaisant toutes les contraintes qui optimise localement WCSS. Cependant, avec les contraintes cannot-link, dans plusieurs cas, l'algorithme n'arrive pas à trouver une solution satisfaisant toutes les contraintes, même lorsqu'une telle solution existe. Nous réalisons les mêmes jeux de test mais pour chaque ensemble de contraintes, COP-kmeans est lancé 1000 fois et nous rapportons le nombre de fois où COP-kmeans trouve une partition. La figure 1 montre le pourcentage des cas réussis lorsque les contraintes cannot-link sont ajoutées. Avec les deux bases Iris et Wine, COP-kmeans ne trouve aucune partition lorsque 150 contraintes sont ajoutées. Notre modèle CP trouve toujours des solutions satisfaisant toutes les contraintes et dans plusieurs cas réussit à prouver l'optimalité des solutions pour la base Iris, comme le montre le tableau 5. La figure 2 montre le résultat lorsque l'on ajoute le même nombre $\#c$ de contraintes must-link et cannot-link. Pour la base Wine, COP-kmeans ne peut trouver aucune partition lorsque $\#c =$

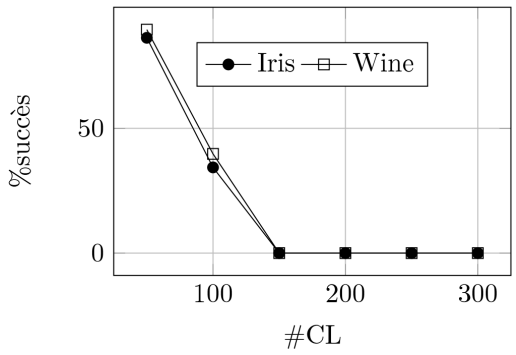


FIGURE 1 – COP-kmeans avec #c cannot-link

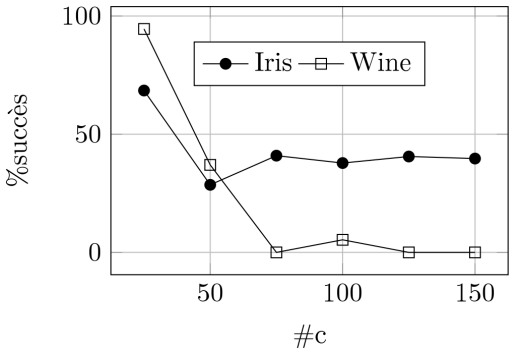


FIGURE 2 – COP-kmeans avec #c ML et #c CL

75 ou 125 ou 150. Notre modèle CP trouve des solutions satisfaisant toutes les contraintes pour tous les cas et trouve un optimum global dans tous les cas pour la base Iris, et dans tous les cas pour la base Wine avec $\#c \geq 100$, comme le montre les tableaux 3 et 6.

6 Conclusion

Des cadres génériques et déclaratifs pour modéliser des tâches de clustering attirent de plus en plus l'attention des communautés de la programmation par contraintes et de la fouille de données. Nous avons développé un cadre basé sur la programmation par contraintes pour le clustering sous contraintes utilisateur avec les critères de diamètre, de la marge ou de la somme des dissimilarités [6, 7, 8]. Dans ce papier, nous développons ce cadre pour intégrer le critère bien connu de la somme des carrés WCSS. Trouver un optimum globale pour ce critère est un problème NP-Difficile et trouver une bonne borne inférieure est aussi difficile [1]. La meilleure approche exacte pour ce critère à notre connaissance est basée sur la programmation linéaire sur des entiers et la génération de colonnes [2]. Une extension de cette approche pour traiter des contraintes utilisateur a été développée [3].

Dans ce papier, nous présentons un calcul de borne inférieure pour la somme des carrés. Nous développons une nouvelle contrainte *wcss* et présentons un algorithme pour filtrer le domaine des variables, qui ne sont pas seulement la variable objective, mais aussi des variables décisionnelles. Des expérimentations sur des bases de données classiques montrent que notre approche obtient une meilleure performance que l'approche basée sur la programmation linéaire sur des entiers et la génération de colonnes.

Nous continuons à améliorer l'algorithme de filtrage pour la contrainte *wcss*, en exploitant les contraintes utilisateur qui sont ajoutées aux tâches de clustering. En effet, les contraintes must-link ajoutées peuvent fractionner les points en composants connexes, ce qui peut aider à améliorer la borne inférieure de la somme des carrés.

Références

[1] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean Sum-of-squares Clustering. *Mach. Learn.*, 75(2) :245–248, May 2009.

[2] Daniel Aloise, Pierre Hansen, and Leo Liberti. An improved column generation algorithm for minimum sum-of-squares clustering. *Mathematical Programming*, 131(1-2) :195–220, 2012.

[3] Behrouz Babaki, Tias Guns, and Siegfried Nijssen. Constrained clustering using column generation. In *Proceedings of the 11th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 438–454, 2014.

[4] Michael Brusco and Stephanie Stahl. *Branch-and-Bound Applications in Combinatorial Data Analysis (Statistics and Computing)*. Springer, 1 edition, July 2005.

[5] M.J. Brusco. An enhanced branch-and-bound algorithm for a partitioning problem. *British Journal of Mathematical and Statistical Psychology*, pages 83–92, 2003.

[6] Thi-Bich-Hanh Dao, Kanh-Chuong Duong, and Christel Vrain. A Declarative Framework for Constrained Clustering. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 419–434, 2013.

[7] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Clustering sous contraintes en ppc. *Revue d'Intelligence Artificielle*, 28(5) :523–545, 2014.

[8] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. *Artificial Intelligence Journal*, To appear.

[9] Ian Davidson and S. S. Ravi. Clustering with Constraints : Feasibility Issues and the k-Means Algorithm. In *Proceedings of the 5th SIAM International Conference on Data Mining*, pages 138–149, 2005.

[10] Ian Davidson and S. S. Ravi. The Complexity of Non-hierarchical Clustering with Instance and Cluster Level Constraints. *Data Mining Knowledge Discovery*, 14(1) :25–61, 2007.

[11] F. Focacci, A. Lodi, and M. Milano. Cost-based domain filtering. In *Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pages 189–203, 1999.

[12] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining : Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[13] Yat Chiu Law and Jimmy Ho-Man Lee. Global constraints for integer and set value precedence. In Mark Wallace, editor, *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, pages 362–376, 2004.

[14] William M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, 66(336) :846–850, 1971.

[15] Jean-Charles Régin. Arc consistency for global cardinality constraints with costs. In *Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pages 390–404, 1999.

[16] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proceedings of the 17th International Conference on Machine Learning*, pages 1103–1110, 2000.

[17] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained K-means Clustering with Background Knowledge. In *Proceedings of the 18th International Conference on Machine Learning*, pages 577–584, 2001.

Clustering conceptuel et relationnel en programmation par contraintes

Thi-Bich-Hanh Dao, Willy Lesaint, Christel Vrain

Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067, Orléans, France
{thi-bich-hanh.dao, willy.lesaint, christel.vrain}@univ-orleans.fr

Résumé

Différentes tâches de clustering existent dans le domaine de la fouille de données. Le clustering conceptuel pour les données qualitatives vise à identifier des concepts, les clusters sont alors définis par ces concepts. Le clustering relationnel cherche à partitionner les données dans des clusters homogènes et/ou bien séparés à partir d'une mesure de dissimilarité entre les objets. Des travaux récents ont proposé des cadres généraux et déclaratifs fondés sur la programmation par contraintes pour le clustering conceptuel ou pour le clustering relationnel. Dans ce papier, nous proposons un cadre plus général qui unifie les clusterings conceptuel et relationnel en programmation par contraintes. Ce cadre permet de tirer avantage des deux approches, en optimisant un critère relationnel ou conceptuel et en intégrant des contraintes qu'elles soient de structure, qu'elles portent sur les distances entre les objets ou sur leurs propriétés communes. Nous proposons de plus un modèle amélioré pour traiter les contraintes conceptuelles reposant sur des contraintes ensemblistes.

1 Introduction

Depuis une dizaine d'années, il a été montré que des formalismes déclaratifs, comme la Programmation par Contraintes, permettaient de modéliser facilement diverses tâches de fouille de données, offrant ainsi une grande flexibilité pour modéliser le problème et intégrer des connaissances utilisateur. En particulier, plusieurs approches se sont intéressées à la classification non supervisée, appelée par la suite clustering, dont le but est de regrouper des objets en classes (clusters) de façon à ce que les objets d'un même cluster soient les plus similaires possibles. On peut distinguer deux approches : le clustering relationnel [2, 3, 4] et le clustering conceptuel [11, 15]. La première, certainement la plus connue, repose sur une mesure de dissimilarité entre les objets, en général la distance euclidienne lorsque les données sont décrites par des attributs numériques. Le regroupement souhaité est souvent modélisé par un critère d'optimisation, comme par exemple minimiser la dissimilarité maximale des objets d'un même cluster, ou maximiser la marge minimale entre les clusters. Cependant, l'interprétation des clusters est difficile, puisque l'importance des différents attributs est agglomérée dans la mesure de dissimilarité.

Le clustering conceptuel s'applique sur des objets décrits par des attributs qualitatifs, souvent booléens. Il vise à regrouper les objets en clusters, de telle manière que chaque cluster puisse être identifié par un ensemble d'attributs communs à tous les objets du cluster. Dans ce cadre, les attributs quantitatifs (numériques) doivent être, dans une étape préalable au clustering, discrétisés en attributs qualitatifs. L'un des intérêts de cette approche est de trouver des clusters qui sont alors des couples formés d'un ensemble d'objets et d'un ensemble d'attributs décrivant ces objets. La discrétisation influence fortement le résultat.

Les méthodes conceptuelles semblent donc plus adaptées aux données symboliques, même s'il est possible de discrétiser des attributs numériques, mais elles ont l'avantage de donner une interprétation des clusters en terme des propriétés que leurs objets vérifient. Quant aux approches relationnelles, elles nécessitent de définir une mesure de dissimilarité entre les objets, ce qui est moins naturel lorsque les données sont hétérogènes, composées de descripteurs quantitatifs et qualitatifs. De plus, elles ne permettent pas d'intégrer directement des contraintes sur les propriétés vérifiées par les objets d'un même cluster.

Nous présentons dans ce papier un cadre général et déclaratif, fondé sur la Programmation par Contraintes, permettant d'exploiter les avantages des deux approches : les données peuvent être décrites

par des attributs quantitatifs ou qualitatifs ; à ces données, sont associés une mesure de dissimilarité calculée sur tout ou partie des attributs et un sous-ensemble des attributs qualitatifs. Il est alors possible d'intégrer des contraintes relationnelles, fondées sur les distances entre les objets et/ou des contraintes conceptuelles fondées sur les propriétés des objets des clusters. Le critère d'optimisation peut lui aussi être relationnel ou conceptuel. Notons que notre approche, lorsqu'appliquée sans contrainte conceptuelle, modélise le clustering relationnel et vice versa. Ce cadre étend donc le clustering relationnel en PPC, introduit par [2, 3, 4] et le clustering conceptuel, présenté dans [11, 15].

De plus, nous proposons dans ce papier deux réalisations des contraintes conceptuelles : la première est l'implantation directe du modèle proposé dans [11], la seconde repose sur des contraintes ensemblistes et nous montrons dans nos expérimentations que cette dernière est plus efficace en termes de propagation que la première. Enfin, nous montrons sur un jeu de données l'intérêt de coupler les approches relationnelle et conceptuelle.

Le papier est organisé comme suit. La section 2 présente le clustering conceptuel, le clustering relationnel et les travaux modélisant ces deux types de clustering dans des approches déclaratives. La section 3 présente notre cadre, intégrant clustering relationnel et conceptuel ; deux réalisations de la vue conceptuelle sont présentées. Enfin, la section 4 est dédiée aux expérimentations.

2 Clustering conceptuel et relationnel sous contraintes utilisateur

Soit $\mathcal{O} = \{o_1, ..., o_n\}$ un ensemble de n objets. Le clustering vise à regrouper les données en un ensemble de clusters homogènes. Le problème est souvent formulé comme la recherche d'une partition en k clusters $\mathcal{C} = \{C_1, .., C_k\}$ satisfaisant les conditions fixées par l'utilisateur et éventuellement, optimisant un critère donné. Pour chaque cluster $c \in \mathcal{C}$, on note O_c l'ensemble des objets du cluster. Les clusters forment une partition, i.e. (1) pour tout $c \in \mathcal{C}$, $O_c \neq \emptyset$, (2) $\cup_c O_c = \mathcal{O}$ et (3) pour tout $c \neq c'$, $O_c \cap O_{c'} = \emptyset$. On distingue en général le clustering conceptuel et le clustering relationnel.

2.1 Clustering conceptuel sous contraintes utilisateur

Dans le clustering conceptuel, les objets sont décrits par un ensemble d'attributs booléens, aussi appelés propriétés, $\mathcal{A} = \{a_1, ..., a_m\}$. Les données sont représentées par une matrice binaire t de taille $n \times m$ telle que $\forall o \in \mathcal{O}, \forall a \in \mathcal{A}$, $t_{oa} = 1$ si et seulement si l'objet o vérifie l'attribut a. Un cluster est alors vu comme un couple formé d'un ensemble d'objets constituant le cluster et d'un ensemble d'attributs caractérisant les objets de ce cluster. Un cluster doit en général vérifier une propriété de fermeture, à savoir les objets du cluster satisfont tous les attributs caractérisant ce cluster (propriété d'intention) et seuls ces objets les vérifient (propriété d'extension). Un cluster satisfaisant cette propriété de fermeture est alos appelé un concept. Pour chaque cluster $c \in \mathcal{C}$, on note A_c l'ensemble des attributs caractérisant le cluster.

Considérons par exemple les données suivantes :

	p_1	p_2	p_3	p_4
o_1	1	1	1	0
o_2	1	1	0	0
o_3	1	1	0	1
o_4	1	0	1	1
o_5	0	1	1	1

Le cluster composé des objets $\{o_1, o_2\}$ satisfait les propriétés p_1 et p_2. Mais il n'est pas fermé, puisque o_3 satisfait aussi les propriétés p_1 et p_2. En revanche les couples $(\{o_1, o_2, o_3\}, \{p_1, p_2\})$ et $(\{o_4, o_5\}, \{p_3, p_4\})$ vérifient la propriété de fermeture et sont des concepts.

Des contraintes et des critères d'optimisation modélisent les conditions qui doivent être satisfaites par les clusters [11, 10].

Contrainte de taille des clusters, aussi appelée contrainte de fréquence. Elle permet de contraindre le nombre d'objets contenus dans chaque cluster par une borne minimale ou maximale, notée b : $\forall c \in \mathcal{C}, |O_c| \lesseqgtr b$.

Contrainte de taille des concepts. La taille d'un cluster, vu comme un concept, est le nombre d'attributs caractérisant ce cluster. On peut de la même manière contraindre ce nombre par une borne b : $\forall c \in \mathcal{C}, |A_c| \lesseqgtr b$.

Contrainte d'extension. L'extension d'un ensemble d'attributs A, $A \subseteq \mathcal{A}$ est l'ensemble des objets vérifiant ces attributs :
$\text{ext}(A) = \{o \in \mathcal{O} \mid \forall a \in A, t_{oa} = 1\}$.

La contrainte d'extension est appliquée à chaque cluster ; elle assure que l'extension des attributs de ce cluster est exactement l'ensemble des objets de ce cluster : $\forall c \in \mathcal{C},\ O_c = \text{ext}(A_c)$.

Notons que $O_c \subseteq \text{ext}(A_c)$ assure que chaque objet du cluster c vérifie tous les attributs du cluster et que $\text{ext}(A_c) \subseteq O_c$ assure qu'aucun autre objet de $\mathcal{O}$ ne possède tous ces attributs.

Etant donné que l'on se place dans le cadre d'une partition des objets, $\text{ext}(A_c) \subseteq O_c$ assure aussi qu'aucun autre cluster ne contient dans son intention cet ensemble d'attributs (sinon ce cluster serait inclus dans

$\text{ext}(A_c)$ et donc dans O_c). Avec cette contrainte, dans ce cadre du clustering conceptuel par partionnement, il est alors impossible d'avoir deux clusters dont l'ensemble des attributs de l'un est inclus dans l'ensemble des attributs de l'autre.

Contrainte d'intention. L'intention d'un ensemble d'objets O, $O \subseteq \mathcal{O}$, est l'ensemble des attributs communs à ces objets :
$\text{int}(O) = \{a \in \mathcal{A} \mid \forall o \in O,\ t_{oa} = 1\}$.

La contrainte d'intention est appliquée à tous les clusters, elle assure pour chaque cluster $c \in \mathcal{C}$ que l'intention des objets de c est exactement l'ensemble des attributs de c : $\forall c \in \mathcal{C}, A_c = \text{int}(O_c)$.

De même que précédemment, notons que $A_c \subseteq \text{int}(O_c)$ assure que chaque objet de c possède bien les attributs A_c du cluster et $\text{int}(O_c) \subseteq A_c$ assure que toutes les propriétés communes à O_c appartiennent bien aux attributs du cluster.

Contrainte de fermeture. La contrainte de fermeture est la conjonction des contraintes d'intention et d'extension : $\forall c \in \mathcal{C},\ O_c = \text{ext}(A_c)\ \wedge\ A_c = \text{int}(O_c)$.

Elle assure que (O_c, A_c) est un concept au sens de l'Analyse Formelle de Concepts.

Maximiser le nombre d'objets par cluster.

Afin d'équilibrer la taille des clusters, on peut choisir d'avoir le plus d'objets possibles dans le cluster de plus petite taille : $\text{maximiser}(\min\{|O_c| \mid c \in \mathcal{C}\})$.

Maximiser le nombre d'attributs caractérisant les clusters.

De la même façon, on peut choisir d'optimiser la taille du concept minimal des clusters : $\text{maximiser}(\min\{|A_c| \mid c \in \mathcal{C}\})$.

2.2 Clustering relationnel sous contraintes utilisateur

On suppose qu'il existe une mesure de dissimilarité entre les objets et on note $d(o, o')$ la dissimilarité entre deux objets $o, o' \in \mathcal{O}$. Cette dissimilarité est calculée à partir des attributs en utilisant, par exemple, la distance euclidienne. Le clustering relationnel regroupe les objets en optimisant un critère fondé sur la dissimilarité entre les objets. Le critère d'optimisation peut être, par exemple, minimiser le diamètre maximal des clusters ou maximiser la marge minimale entre clusters.

Minimiser le diamètre. Minimiser le diamètre consiste à minimiser le diamètre du cluster le plus grand, c'est-à-dire à minimiser la plus grande distance entre deux objets d'un même cluster :

$\text{minimiser}(\max\{d(o, o') \mid c \in \mathcal{C} \wedge o \in O_c \wedge o' \in O_c\})$.

Notons que minimiser le diamètre a pour effet de séparer dans des clusters différents les objets éloignés.

Maximiser la marge. Maximiser la marge propose une vision duale de l'optimisation précédente. En effet, on souhaite ici différencier au maximum les clusters, en maximisant la distance entre des points situés dans des clusters différents.

$\text{maximiser}(min\{d(o, o') \mid c \neq c' \in \mathcal{C}, o \in O_c, o' \in O_{c'}\})$.

Des contraintes utilisateur sont également intégrées dans les tâches de clustering relationnel. Elles peuvent être des contraintes sur des objets (must-link, cannot-link) [17, 18] ou des contraintes sur des clusters [5, 6].

Contraintes must-link et cannot-link. Une contrainte must-link sur deux objets o, o' indique qu'ils doivent être dans le même cluster : $\forall c \in \mathcal{C},\ o \in O_c \Leftrightarrow o' \in O_c$. A l'inverse, une contrainte cannot-link indique que deux objets ne doivent pas être dans le même cluster : $\forall c \in \mathcal{C},\ \neg(o \in O_c \wedge o' \in O_c)$.

Contrainte de taille des clusters. Cette contrainte est identique à la contrainte du même nom (ou contrainte de fréquence) dans le clustering conceptuel. Elle permet de fixer une valeur maximale ou minimale sur le nombre d'objets dans chaque cluster.

Contrainte de diamètre La contrainte de diamètre permet de fixer une borne supérieure b au diamètre des clusters, i.e., à la distance maximale entre les objets du cluster : $\forall c \in \mathcal{C}, \forall o, o' \in O_c,\ d(o, o') \leq b$.

Contrainte de marge. On peut également contraindre la distance entre les clusters en imposant que cette distance soit supérieure à un seuil b : $\forall c \neq c' \in \mathcal{C}, \forall o \in O_c, \forall o' \in O_{c'},\ d(o, o') \geq b$.

2.3 Travaux connexes

Depuis quelques années, différents travaux investissent les cadres génériques et déclaratifs pour différentes tâches de clustering sous contraintes utilisateur. Ces cadres sont développés à base de la programmation par contraintes, de la programmation linéaire sur des entiers [16, 9, 1] ou de la programmation quadratique [19]. L'objectif général est de proposer un cadre où l'utilisateur peut intégrer différents types de contraintes.

Concernant le clustering conceptuel, les premiers travaux modélisent la recherche d'itemsets (ensembles d'attributs) dans le cadre de la programmation par contraintes [8, 10], de SAT [15, 12] ou d'ASP [13]. La recherche d'itemsets est généralisée en recherche d'ensembles d'itemsets, formalisant ainsi le clustering conceptuel. Dans [11], les auteurs reformulent les problèmes de recherche d'ensembles d'items en terme de programmation par contraintes. Ils formalisent ainsi la plupart des contraintes classiques de fouille de données. Le cadre déclaratif et très général de la programmation par contrainte permet alors de s'attaquer à la

plupart des problèmes de recherche d'itemsets. Le clustering conceptuel est formulé par la recherche d'un ensemble d'itemsets satisfaisant des contraintes spécifiques [11]. Dans cette lignée, un langage basé sur des contraintes a été proposé [15]. Des tâches de clustering conceptuel peuvent être formulées par des requêtes dans ce langage, qui sont ensuite traduites par des clauses SAT puis résolues par un solveur SAT.

Quant au clustering relationnel à base de dissimilarités, plusieurs cadres génériques sont développés permettant à l'utilisateur d'intégrer différents types de contraintes. Nous pouvons citer par exemple l'approche à base de programmation linéaire sur des entiers pour le critère de la somme des carrés intra-cluster [1] ou l'approche à base de SAT pour le cas de partition en 2 clusters ($k = 2$) [7]. Basée sur la programmation par contraintes, un cadre général et déclaratif a été proposé [2, 3, 4]. Ce cadre offre à l'utilisateur le choix de différents critères d'optimisation et des combinaisons de différents types de contraintes utlisateur.

3 Modélisation par la programmation par contraintes

Nous présentons un cadre en programmation par contraintes, qui unifie le clustering conceptuel et le clustering relationnel sous contraintes utilisateur. Ce cadre permet de traiter des données hétérogènes, qui peuvent être caractérisées par des attributs numériques (quantitatifs) et des attributs symboliques (qualitatifs). L'utilisateur peut choisir parmi les attributs ceux qui seront étudiés comme propriétés des objets et ceux utilisés pour calculer une dissimilarité entre paires d'objets. Il peut ensuite poser des contraintes conceptuelles sur des propriétés ainsi que des contraintes relationnelles fondées sur la dissimilarité. Il peut choisir de trouver toutes les solutions ou de trouver une solution optimale, étant donné un critère d'optimisation, pouvant relever du clustering conceptuel ou du clustering relationnel. Le cadre complet permet donc plusieurs possibilités :

- il peut se décliner en des tâches du clustering purement conceptuel ou purement relationnel,
- il permet de réaliser des tâches du clustering conceptuel (ou relationnel) en intégrant des contraintes relationnelles (ou conceptuelles, respectivement).

Nous présentons ce cadre en quatre parties : contraintes liées à la structure de l'ensemble de clusters, contraintes relationnelles, contraintes conceptuelles et critères d'optimisation. Le modèle est basé sur le modèle du clustering relationnel sous contraintes utilisateur [3, 4]. Nous présentons deux réalisations pour les contraintes conceptuelles : une réalisation binaire qui est une extension directe des travaux de [8, 10] et une réalisation utilisant des contraintes ensemblistes.

3.1 Contraintes sur la structure

Dans ce qui suit, on considère un ensemble $\mathcal{O}$ de n objets décrits sur un ensemble d'attributs booléens $\mathcal{A}$ de m attributs. Sans perdre de généralité, nous supposons que les objets et les attributs sont indexés et identifiés par leur indice, qui est $o \in [1, n]$ [1] pour les objets et $a \in [1, m]$ pour les attributs. Ceci implique aussi que $\mathcal{O}$ (ou $\mathcal{A}$) peut être utilisé pour l'ensemble $[1, n]$ (ou $[1, m]$, resp.) ou vice versa. La description des objets est représentée par une matrice binaire t de taille $n \times m$ telle que $\forall o \in \mathcal{O}, \forall a \in \mathcal{A}$, $t_{oa} = 1$ si et seulement si l'objet o vérifie l'attribut a. De plus, une mesure de dissimilarité d permet de calculer une dissimilarité entre toute paire d'objets.

Chaque cluster est identifié par son numéro, qui varie de 1 à k. Soit $\mathcal{C}$ l'ensemble des clusters $[1, k]$. Pour représenter l'appartenance de chaque objet $o \in [1, n]$ aux clusters, nous utilisons, comme dans [3], des variables $G : \mathcal{O} \to \mathcal{C}$, telles que $G[o] = c$ signifie que l'objet o appartient au cluster c. Le domaine des variables $G[o]$ est l'ensemble des entiers $[1, k]$.

Contraintes de partition Le choix des variables G représente naturellement une partition. Cependant il peut engendrer des solutions symétriques. Par exemple, deux solutions dans lesquelles deux clusters sont intervertis sont symétriques. Les objets composant les clusters sont les mêmes mais les numéros des clusters diffèrent. Supprimer ces symétries se fait par l'ajout de contraintes. Une stratégie consiste à numéroter les clusters dans l'ordre croissant. Plus précisément, le premier objet est placé dans le cluster numéro 1 puis les suivants sont placés peu à peu soit dans un cluster déjà créé, soit dans un nouveau cluster. Le numéro donné au nouveau cluster doit être égal au plus grand numéro déjà utilisé plus 1. Si l'on considère le tableau G, on doit avoir $G[1] = 1$ et pour toute valeur $c \in [2, k]$, s'il existe un indice i tel que $G[i] = c$, alors il doit exister un indice $j < i$ tel que $G[j] = c - 1$. Ces conditions sont exprimées par la contrainte $\mathit{precede}(G, [1, \ldots, k])$ [14].

Pour assurer d'obtenir à la fin k clusters non vides, la valeur k doit apparaître au moins une fois dans le tableau G. Ceci est assuré par la contrainte : $\mathit{atleast}(1, G, k)$.

1. Nous utilisons la notation $[1, n]$ pour désigner l'ensemble des entiers $\{1, .., n\}$.

Contraintes must-link et cannot-link Une contrainte must-link sur deux objets o, o' se traduit facilement par : $G[o] = G[o']$. De la même façon, une contrainte cannot-link se traduit par : $G[o] \neq G[o']$.

Contrainte de taille de clusters Le fait que chaque cluster c doit avoir au moins (ou au plus) b objets revient à ce que la valeur c apparaisse au moins (resp. au plus) b fois dans le tableau G. Ceci est représenté par la contrainte $atleast(b, G, c)$ (resp. $atmost(b, G, c)$), pour toute valeur $c \in \mathcal{C}$.

3.2 Contraintes relationnelles

Le diamètre maximal des clusters et la marge minimale entre clusters sont représentés respectivement par des variables D et S. Leur domaine est l'intervalle formé par la distance minimale et la distance maximale entre deux objets.

Contrainte de diamètre Une contrainte de diamètre fixe une borne supérieure $d_{\max}$ au diamètre des clusters. Tous les objets $o, o' \in \mathcal{O}$ tels que $d(o, o') > d_{\max}$ doivent être placés dans des clusters différents. Donc pour tout $o, o' \in \mathcal{O}$, si $d(o, o') > d_{\max}$, on pose une contrainte $G[o] \neq G[o']$.

Contrainte de marge Une contrainte de marge fixe une borne inférieure $d_{\min}$ à la marge entre clusters. Tous les objets $o, o' \in \mathcal{O}$ tels que $d(o, o') < d_{\min}$ doivent être placés dans le même cluster. Donc pour tout $o, o' \in \mathcal{O}$, si $d(o, o') < d_{\min}$, on pose une contrainte $G[o] = G[o']$.

3.3 Contraintes conceptuelles

Chaque cluster, composé d'un ensemble d'objets, est aussi caractérisé par un ensemble d'attributs. Pour représenter les attributs associés aux clusters et modéliser les contraintes conceptuelles permettant d'exprimer des conditions sur ces propriétés, nous présentons deux modèles différents. Le premier modèle utilise celui de [10, 11], à base de variables binaires. Le deuxième modèle repose sur des variables ensemblistes.

3.3.1 Modèle à base de variables binaires

Pour représenter les attributs associés aux clusters, de même que dans le modèle de [11], nous utilisons des variables binaires $A : \mathcal{C} \times \mathcal{A} \to \{0, 1\}$, où $A[c, a] = 1$ signifie que l'attribut a participe à A_c, l'ensemble qui caractérise le cluster c. Cela nécessite $k \times m$ variables binaires.

Contrainte de taille de concepts Elle peut se représenter par :

$$\forall c \in \mathcal{C}, \sum_{a \in \mathcal{A}} A[c, a] \lesseqgtr b \tag{1}$$

où b est la borne de taille. Cette contrainte est réalisée par k contraintes de somme linéaire sur des variables de A.

Contrainte d'extension Pour tout $o \in \mathcal{O}$, pour tout $c \in \mathcal{C}$:

$$G[o] = c \Leftrightarrow \sum_{1 \leq a \leq m} A[c, a](1 - t_{oa}) = 0 \tag{2}$$

Cette contrainte est réalisée par $n \times k$ contraintes réifiées, chacune est liée à une contrainte de somme linéaire sur des variables de A.

Contrainte d'intention Pour tout $c \in \mathcal{C}$, pour tout $a \in \mathcal{A}$:

$$A[c, a] \Leftrightarrow \sum_{1 \leq o \leq n} (G[o] = c)(1 - t_{oa}) = 0 \tag{3}$$

Cette contrainte est réalisée par $m \times k$ contraintes réifiées, chacune est liée à une contrainte de somme linéaire sur des variables booléennes représentant $G[o] = c$.

Contrainte de fermeture Par sa définition, la contrainte de fermeture sur $\mathcal{C}$ sera réalisée par les contraintes (2) et (3), qui représentent respectivement l'extension et l'intention des clusters de $\mathcal{C}$.

3.3.2 Modèle utilisant des variables ensemblistes

Nous proposons d'utiliser des variables ensemblistes $E : \mathcal{C} \to \mathcal{P}(\mathcal{A})$ pour représenter les ensembles d'attributs associés aux clusters. Cela nécessite donc k variables ensemblistes E_c, où chaque variable représente un ensemble $E_c \subseteq \mathcal{A}$.

De la même manière, la représentation des données sur les attributs qualitatifs est modifiée : chaque objet $o \in \mathcal{O}$ est caractérisé par l'ensemble t_o des attributs qu'il vérifie.

Avec cette modélisation, la réalisation des contraintes suivantes est modifiée.

La contrainte de taille de concept (1) devient :

$$\forall c \in \mathcal{C},\ |E[c]| \lesseqgtr b$$

qui est représentée par une contrainte *cardinality* pour chaque $c \in \mathcal{C}$.

La contrainte extension (2) devient

$$\forall c \in \mathcal{C}, \forall o \in \mathcal{O},\ G[o] = c \Leftrightarrow E_c \subseteq t_o$$

Cette contrainte est réalisée par $n \times k$ contraintes réifiées, chacune est liée à une contrainte d'inclusion.

La contrainte intention (3) devient

$$\forall c \in \mathcal{C},\ E_c = \cap_{G[o]=c} t_o$$

Elle est réalisée par k contraintes $E_c = \cap_{G[o]=c} t_o$, avec $c \in \mathcal{C}$. Chacune de ces contraintes nécessite n contraintes de domaine *dom* réifiées pour construire l'ensemble $I_c = \{o \in \mathcal{O} \mid G[o] = c\}$ et une contrainte *element* ensembliste effectuant $E_c = \cap(\langle t_1, ..., t_n \rangle [I_c])$.

3.4 Critères d'optimisation

Minimiser le diamètre Pour représenter le diamètre maximal des clusters nous utilisons une variable D du domaine des nombres flottants. Par conséquent, si deux objets o, o' ont une dissimilarité plus grande que D, ils doivent être dans des clusters différents. Cela est représenté par des contraintes réifiées. Nous posons donc pour chaque paire $o < o' \in \mathcal{O}$ la contrainte réifiée :

$$d(o, o') > D \Rightarrow G[o] \neq G[o'].$$

Lorsque l'utilisateur fait le choix de minimiser le diamètre, cela revient à minimiser D. Cependant, le nombre de contraintes réifiées est quadratique par rapport au nombre d'objets et la gestion des contraintes réifiées nécessite des variables et des contraintes supplémentaires. Avec la minimisation de D par *branch-and-bound*, nous remplaçons ces contraintes par :

- A chaque solution intermédiaire Δ, calculer le diamètre $D(\Delta)$.
- Pour chaque paire $o < o' \in \mathcal{O}$ telle que $d(o, o') > D(\Delta)$, poser la contrainte $G[o] \neq G[o']$.

Maximiser la marge La marge minimale entre clusters est représentée par une variable S du domaine des nombres flottants. De la même façon que pour minimiser le diamètre, lorsque l'utilisateur souhaite maximiser la marge entre clusters, la valeur de la variable S est maximisée. Puisque S représente la marge minimale entre clusters, deux objets $o, o' \in \mathcal{O}$ vérifiant $d(o, o') < S$ doivent être dans le même cluster. La minimisation de S se réalise par :

- A chaque solution intermédiaire Δ, calculer la marge $S(\Delta)$.
- Pour chaque paire $o < o' \in \mathcal{O}$ telle que $d(o, o') < S(\Delta)$, poser la contrainte $G[o] = G[o']$.

Maximiser le nombre d'objets par cluster Si l'utilisateur choisit de maximiser le nombre d'objets par cluster, une variable F qui représente la fréquence des clusters est introduite. Son domaine est l'ensemble $[1, n]$. Pour chaque valeur $c \in \mathcal{C}$, nous posons une contrainte *atleast*(F, G, c) pour indiquer que F est la fréquence du cluster c. La valeur de F est maximisée.

Maximiser le nombre d'attributs caractérisant les clusters Dans ce cas, une variable T du domaine $[1, m]$ est introduite. Les contraintes suivantes permettent d'associer T au nombre minimal d'attributs des clusters, dans le cas du modèle avec des variables binaires A :

$$\forall c \in \mathcal{C},\ T \leq \sum_{a \in \mathcal{A}} A[c, a]$$

ou dans le cas du modèle avec des variables ensemblistes E :

$$\forall c \in \mathcal{C},\ T \leq |E[c]|$$

La valeur de T est maximisée.

4 Expérimentations

Les modèles présentés précédemment ont été implantés en utilisant la version 4.2.1 de la bibliothèque de programmation par contrainte Gecode [2]. Les expérimentations suivantes ont deux objectifs : comparer l'efficacité des modèles binaire et ensembliste et montrer l'intérêt de notre cadre unifié.

Nous testons notre approche sur la base *Automobile* disponible dans le répertoire UCI [3]. Après retrait des données incomplètes, la base contient 193 objets (des modèles d'automobiles) et 23 attributs. 22 attributs (9 symboliques et 13 réels) concernent des caractéristiques techniques des véhicules. Par exemple, une automobile est caractérisée par un type de motorisation (diesel ou essence), des roues motrices (4 roues motrices, 2 à l'avant ou 2 à l'arrière), par une puissance (comprise entre 48 to 288),... Le dernier attribut est le prix (ceux-ci vont de 5118 à 45400).

Nous choisissons de prendre ce prix pour calculer la dissimilarité entre les véhicules, la distance entre chaque véhicule étant la différence de prix. Cela permet d'obtenir des résultats aisément interprétables. Les caractéristiques techniques sont quant à elles discrétisées et fournissent 64 attributs pour le côté conceptuel (pour chaque donnée réelle, deux attributs sont créés pour répartir les objets selon leur position par rapport à la médiane).

Afin d'exploiter le partitionnement des objets, le branchement est d'abord effectué sur les objets (G) en les prenant par ordre croissant, puis sur les attributs (A ou E selon le modèle).

4.1 Comparaison des modèles binaire et ensembliste

La différence entre les modèles binaire et ensembliste porte sur les contraintes conceptuelles. Les tests sont

2. `http://www.gecode.org`
3. `http://archive.ics.uci.edu/ml`

Contr.	k	Modèle bin.		Modèle ens.	
		time(s)	#nodes	time(s)	#nodes
int.	2	0.10	2338	0.14	2877
	3	0.63	15109	0.55	13986
ext.	2	0.03	79	0.03	52
	3	8.40	145098	0.08	979
	4	6960	>1M	0.55	8636
	5	_	_	2.49	36264
ferm.	2	0.03	24	0.03	47
	3	13.50	72494	0.15	963
	4	9900	>1M	1.29	8585
	5	_	_	5.47	36057

TABLE 1 – Approches binaire et ensembliste

Test	Contrainte	Optimisation	#cl.
(a)	fermeture	#obj./cl.	2
(b)	fermeture	#att./cl.	2
(c)	intention	diamètre	2
(d)	fermeture	diamètre	2
(e)	fermeture	#obj./cl.	3
(f)	fermeture	#att./cl	3
(g)	intention	diamètre	3
(h)	fermeture	diamètre	3

TABLE 2 – Tests

donc effectués respectivement avec la contrainte d'intention, la contrainte d'extension et enfin la conjonction des deux (i.e. la fermeture). L'optimisation est réalisée en minimisant le diamètre du plus grand cluster. Les performances obtenues sont résumées dans la table 1 (le temps de calcul est limité à 3h, le caractère _ indique que ce temps de calcul est dépassé).

Les résultats mettent clairement en évidence l'efficacité du modèle ensembliste par rapport au modèle binaire. Quel que soit le modèle choisi, la contrainte d'intention seule ne permet pas de réduire suffisamment l'espace de recherche et se révèle peu efficace. La différence entre les deux modèles porte sur la contrainte d'extension (que l'on retrouve également dans la fermeture). L'utilisation de contraintes réifiées liées à des contraintes linéaires dans le cas binaire est très pénalisante. Au contraire, les contraintes d'inclusion du modèle ensembliste permettent une bonne propagation et un nombre de nœuds à explorer beaucoup moins important.

4.2 Comparaison avec des approches conceptuelles ou relationnelles

Les tests suivants sont réalisés avec le modèle ensembliste. L'objectif est d'obtenir des concepts suivant le prix avec 2 ou 3 clusters afin que les résultats obtenus soient les plus interprétables possible. Plusieurs types de tests (listés dans la table 2) sont réalisés : avec une approche purement conceptuelle, avec une approche purement relationnelle et enfin en utilisant notre cadre unifié.

Expérimentations avec 2 clusters Les premières expérimentations sont réalisées avec 2 clusters, les tailles des clusters obtenus sont données dans la table 3 et la figure 1 indique la répartition des objets dans les clusters en fonction de leur prix.

	Clusters				
test	#obj./cl.		#att./cl.		diam.
(a)	95	98	2	2	40282
(b)	94	99	2	2	39901
(c)	176	17	1	7	19848
(d)	98	95	1	2	37387

TABLE 3 – Taille des clusters pour k=2

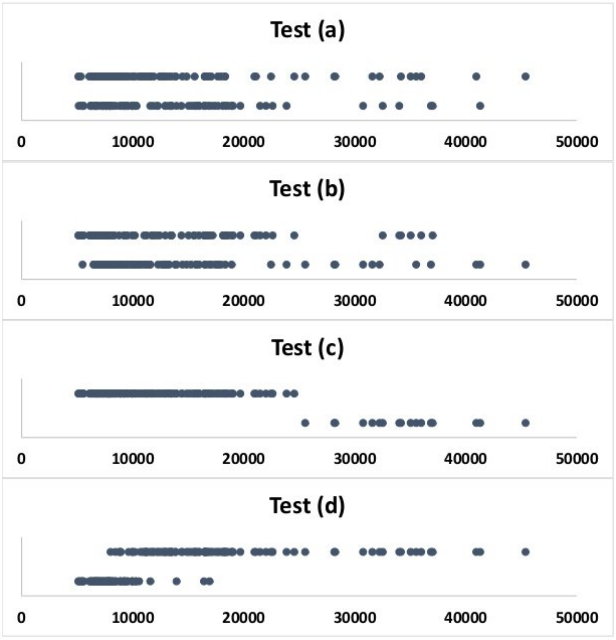


FIGURE 1 – Répartition des objets/cluster pour k=2

Cadre conceptuel : les tests (a) et (b) sont réalisés avec la contrainte de fermeture en maximisant respectivement la taille des clusters (a) et la taille des concepts (b). La distance n'intervenant pas, on se place donc ici dans un cadre purement conceptuel. Pour les deux tests, la taille des clusters est équilibrée. Alors qu'on aurait pu espérer que les prix soient corrélés aux caractéristiques techniques, les clusters obtenus ne représentent pas de regroupement de véhicules par prix. L'intégration des prix dans les attributs conceptuels n'est pas présentée ici car elle ne modifie pas les clusters obtenus.

Cadre relationnel : le test (c) est réalisé en minimisant le diamètre et en utilisant la contrainte d'intention pour obtenir les attributs communs aux objets de chaque cluster. En se plaçant ainsi dans un cadre purement relationnel, le diamètre est forcément optimal et les intervalles de prix des clusters totalement disjoints. On obtient une bonne caractérisation du second cluster par 7 attributs (roues motrices à l'arrière, poids élevé, moteur puissant, forte consommation...) du fait de sa petite taille. Au contraire, le premier cluster n'est lui caractérisé que par 1 attribut (moteur à l'avant). La différence de taille des clusters s'explique par des différences de prix très importantes dans le haut de gamme. Notez qu'utiliser l'ensemble des attributs dans le calcul de dissimilarité ne permet pas d'équilibrer davantage la taille des clusters.

Cadre unifié : en remplaçant la contrainte d'intention par la contrainte de fermeture, le test (d) permet d'obtenir des concepts tout en minimisant le diamètre. La taille des clusters obtenus est équilibrée même si les concepts contiennent peu d'attributs (moteur à l'avant et poids faible pour l'un, poids élevé pour l'autre), ce qui est logique étant donné le nombre de clusters et l'utilisation de la contrainte de fermeture. La répartition des objets fait apparaître que le premier cluster correspond aux véhicules à bas coût et le second aux véhicules plus onéreux. La différence de diamètre entre les deux clusters s'explique, comme précédemment par des écarts de prix plus importants dans les modèles haut de gamme. L'extension permet ici de mettre en évidence le poids comme attribut discriminant.

Expérimentations avec 3 clusters Passer de deux à trois clusters permet de réduire leur taille et par conséquent d'augmenter la taille des concepts. Les résultats des expérimentations pour 3 clusters sont résumés dans la table 4 et la figure 2.

Cadre conceptuel : les tests (e) et (f) sont réalisés avec la contrainte de fermeture en maximisant respectivement le nombre d'objets et le nombre d'attributs (on se place donc à nouveau dans un cadre purement conceptuel). Les résultats obtenus par les 2 méthodes

	Clusters						
test	#objets			#attributs			diam.
(e)	63	66	64	3	3	3	38615
(f)	68	61	64	3	3	3	38615
(g)	161	20	12	1	5	10	13226
(h)	72	108	13	4	2	5	33550

TABLE 4 – Taille des clusters pour k=3

sont très proches, que ce soit au niveau de la taille des clusters et des concepts, qu'au niveau du contenu de ces concepts (seul un attribut varie : le poids du véhicule est remplacé par sa hauteur). Cependant dans ces deux cas encore, comme pour les tests sur deux clusters, la répartition selon les prix est peu probante.

Cadre relationnel : le test (g) réalisé comme pour (c) dans un cadre purement relationnel fournit toujours un diamètre optimal et des objets parfaitement répartis par prix. Cependant, la taille des clusters est toujours très déséquilibrée.

Cadre unifié : le test (h) permet d'obtenir des clusters de tailles plus équilibrées que l'approche relationnelle et reflétant davantage les gammes d'automobile que l'approche conceptuelle. Le premier cluster caractérise 72 véhicules par 4 attributs : un poids élevé, un moteur de grande taille, une puissance élevée et une consommation sur autoroute importante. Le second cluster contient 108 véhicules caractérisés par 2 attributs : un moteur à l'avant et une consommation sur autoroute faible. Pour le dernier cluster, les 13 véhicules sont caractérisés par 5 attributs : un moteur essence, situé à l'avant, de petite taille, avec 4 cylindres et une consommation sur autoroute élevée. Le premier cluster correspond clairement aux véhicules haut de gamme alors que les deux derniers clusters contiennent des véhicules de prix plus bas.

Intérêt du cadre unifié Les résultats obtenus précédemment démontrent l'intérêt d'une méthode unifiée qui tire parti des points forts des approches conceptuelle et relationnelle. Les tests (i) et (j) montrent comment améliorer encore les résultats grâce à l'apport des contraintes utilisateur et à une exploitation plus poussée de la dissimilarité. Les résultats obtenus se trouvent dans la table 5 et la figure 3.

Le test (i) est réalisé dans le cas d'une recherche de 3 clusters en optimisant le diamètre, avec la contrainte de fermeture et en ajoutant deux contraintes utilisateur : une taille de clusters supérieure ou égale à 40 et une taille de concepts supérieure ou égale à 2. Par rapport au test (h) réalisé sans ces contraintes utilisateur, il permet, moyennant une augmentation des diamètres

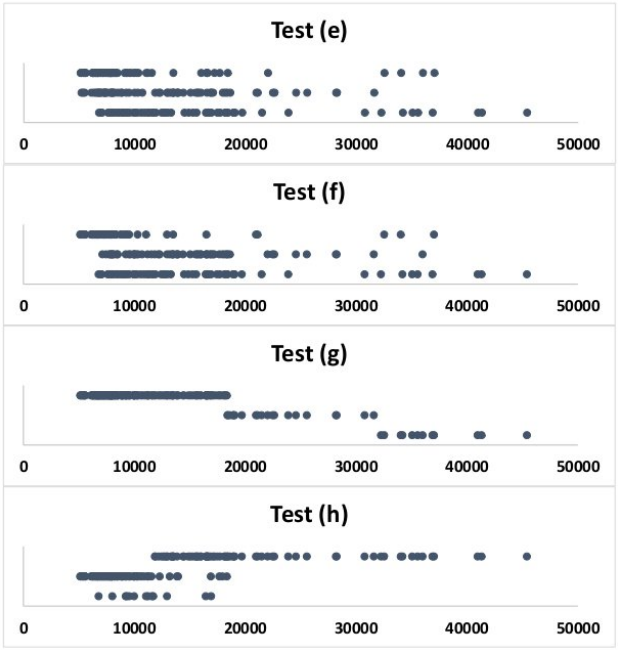


FIGURE 2 – Répartition des objets/cluster pour k=3

des clusters, d'obtenir des tailles de clusters plus équilibrées. Le premier cluster caractérise les véhicules avec un poids et un ratio de compression important. Les véhicules du second cluster sont caractérisés par un poids faible et un moteur à l'avant. Enfin les caractéristiques du dernier cluster sont : un moteur essence, situé à l'avant, un poids important et un ratio de compression faible.

Enfin, notre cadre unifié n'est pas restreint à des dissimilarités calculées sur un seul attribut. Pour notre exemple, on peut considérer qu'un véhicule haut de gamme est à la fois cher et puissant. Le test (j) est ainsi réalisé en calculant une dissimilarité prenant en compte à la fois la puissance et le prix du véhicule. En utilisant les mêmes contraintes que précédemment, les résultats obtenus font apparaître des objets mieux répartis dans les clusters et des concepts bien définis. Le premier cluster caractérise les véhicules haut de gamme par les attributs essence, taille de moteur élevée, système d'injection mpfi, course du piston faible, forte consommation urbaine et sur autoroute. Les véhicules du second cluster sont caractérisés par un moteur à l'avant et de grande taille ainsi qu'une course du piston élevée. Enfin, le dernier cluster regroupe des véhicules de bas de gamme caractérisés par un moteur positionné à l'avant et de petite taille.

test	Clusters						
	#objets			#attributs			diam.
(i)	57	95	41	2	2	4	36479
(j)	42	56	95	7	3	2	38411

TABLE 5 – Taille des clusters pour (i) et (j)

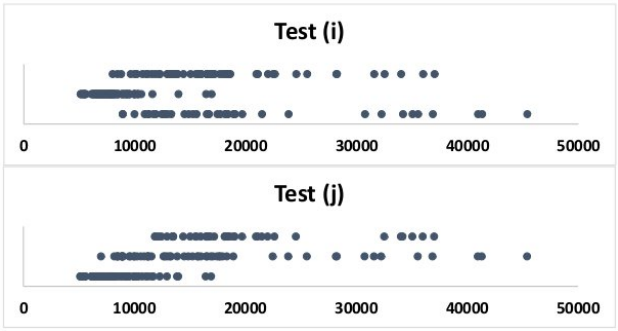


FIGURE 3 – Répartition des objets/cluster pour les tests (i) et (j)

5 Conclusion

Dans cet article, nous avons proposé un cadre à base de programmation par contraintes, qui unifie le clustering relationnel et le clustering conceptuel. Ce cadre complet offre différentes possibilités, car il peut se décliner en des tâches de clustering purement conceptuel ou purement relationnel, ou permettre de réaliser des tâches de clustering conceptuel (ou relationnel) en intégrant à la fois des contraintes de structure, relationnelles ou conceptuelles. Nous avons présenté deux réalisations pour les contraintes conceptuelles : une réalisation binaire utilisant des variables et contraintes booléennes et une réalisation utilisant des variables et contraintes ensemblistes. Enfin, des exemples concrets ont démontré l'intérêt de ce cadre unifié pour l'utilisateur.

Nous continuons à développer ce cadre général dans plusieurs directions. Le choix du nombre k de clusters est pour l'instant une contrainte dure du modèle. Le passage de k en variable du problème implique l'utilisation d'une fonction objective plus complexe que celles proposées ici. En effet, minimiser le diamètre, maximiser la marge ou maximiser la taille des concepts pousse k vers des valeurs élevées. Au contraire, maximiser la taille des clusters tend à amener k vers des valeurs faibles. Nous souhaitons également améliorer l'efficacité de notre cadre pour pouvoir traiter des bases de données de plus grandes dimensions. Enfin, nous souhaitons étendre le modèle par l'intégration de nouvelles contraintes conceptuelles.

Références

[1] Behrouz Babaki, Tias Guns, and Siegfried Nijssen. Constrained clustering using column generation. In *CPAIOR*, pages 438–454, 2014.

[2] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. A Declarative Framework for Constrained Clustering. In *ECMLPKDD*, pages 419–434, 2013.

[3] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Clustering sous contraintes en PPC. *Revue d'Intelligence Artificielle*, 28(5) :523–545, 2014.

[4] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. *Artificial Intelligence*, To appear.

[5] Ian Davidson and S. S. Ravi. Clustering with Constraints : Feasibility Issues and the k-Means Algorithm. In *ICDM*, pages 138–149, 2005.

[6] Ian Davidson and S. S. Ravi. The Complexity of Non-hierarchical Clustering with Instance and Cluster Level Constraints. *Data Min. Knowl. Disc.*, 14(1) :25–61, 2007.

[7] Ian Davidson, S. S. Ravi, and Leonid Shamis. A SAT-based Framework for Efficient Constrained Clustering. In *ICDM*, pages 94–105, 2010.

[8] Luc De Raedt, Tias Guns, and Siegfried Nijssen. Constraint programming for itemset mining. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 204–212, 2008.

[9] Sean Gilpin, Siegfried Nijssen, and Ian N. Davidson. Formalizing hierarchical clustering as integer linear programming. In *AAAI*, pages 372–378, 2013.

[10] Tias Guns, Siegfried Nijssen, and Luc De Raedt. Itemset mining : A constraint programming perspective. *Artificial Intelligence*, 175 :1951–1983, 2011.

[11] Tias Guns, Siegfried Nijssen, and Luc De Raedt. k-Pattern set mining under constraints. *IEEE Transactions on Knowledge and Data Engineering*, 25(2) :402–418, 2013.

[12] Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. The top-k frequent closed itemset mining using top-k SAT problem. In *ECMLPKDD*, pages 403–418, 2013.

[13] Matti Järvisalo. Itemset mining as a challenge application for answer set enumeration. In *LPNMR*, pages 304–310, 2011.

[14] Yat Chiu Law and Jimmy Ho-Man Lee. Global constraints for integer and set value precedence. In Mark Wallace, editor, *CP*, pages 362–376, 2004.

[15] Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni. Constrained Clustering Using SAT. In *IDA*, pages 207–218, 2012.

[16] Marianne Mueller and Stefan Kramer. Integer Linear Programming Models for Constrained Clustering. In *DS*, pages 159–173, 2010.

[17] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *ICML*, pages 1103–1110, 2000.

[18] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained K-means Clustering with Background Knowledge. In *ICML*, pages 577–584, 2001.

[19] Xiang Wang, Buyue Qian, and Ian Davidson. On constrained spectral clustering and its applications. *Data Min. Knowl. Discov.*, 28(1) :1–30, 2014.

Etude de modèles de programmation par contraintes pour le problème du voyageur de commerce avec fenêtres de temps

Sylvain Ducomman[1 2] **Hadrien Cambazard**[1] **Bernard Penz**[1]

[1] Univ. Grenoble Alpes, G-SCOP, F-38000 Grenoble, France
CNRS, G-SCOP, F-38000 Grenoble, France
[2] Geoconcept SA, 92220 Bagneux, France
{prénom.nom}@grenoble-inp.fr sylvain.ducomman@geoconcept.com

Résumé

Le problème du voyageur de commerce avec fenêtres de temps (TSPTW) consiste à visiter un ensemble de clients, en respectant pour chaque client une fenêtre de temps donnée, tout en minimisant la distance totale parcourue. Ce problème est une extension du problème du voyageur de commerce qui est NP-difficile au sens fort. Il a été largement traité, entre autres par la Programmation Par Contraintes (CP). Dans cet article, nous présentons différents modèles par Programmation Par Contraintes. Nous proposons ensuite différentes approches dans le but d'améliorer la propagation, notamment en utilisant la contrainte globale WeightedCircuit. Enfin, nous étudions plusieurs stratégies de branchement ainsi qu'une technique simple d'enregistrement de nogood. L'évaluation expérimentale est réalisée sur des benchmarks de la littérature. Ce travail est une première étape pour évaluer des techniques de filtrage basées sur les coûts et traiter des problèmes de tournées de véhicules avec fenêtres de temps.

Abstract

The Traveling Salesman Problem with time-windows (TSPTW) is to visit a set customers within given time-windows while minimizing the overall distance of the tour. It is a widely studied extension of the Traveling Salesman Problem and has received some attention of the Constraint Programming (CP) community in the past. We investigate in this paper several CP models for the problem. In particular, we evaluate the interest of the WeightedCircuit global constraint. We also discuss branching strategies and a simple nogood recording technique. Results are presented on a well known benchmark of the TSPTW and aims at understanding the current performances of CP technology on this problem. This is a first step in a study of cost-based filtering techniques for routing problems with time-windows.

1 Introduction

Le Problème du Voyageur de Commerce (TSP) est un problème combinatoire largement étudié. Le problème rencontré par ce voyageur est de visiter chacun de ses clients une et une seule fois avant de revenir chez lui en parcourant une distance minimale. Dans cette étude, nous nous intéressons à une extension du TSP prenant en compte les fenêtres de temps et le temps de service des clients. Il s'agit du Problème du Voyageur du Commerce avec Fenêtres de Temps (TSPTW). L'objet du TSPTW est de trouver le plus court chemin visitant les clients une et une seule fois tout en respectant leurs fenêtres de temps. Cela signifie que les clients doivent être visités dans l'intervalle correspondant. Il est possible d'arriver chez un client avant sa date de disponibilité et d'attendre pour effectuer le service au début de sa fenêtre de temps. Le TSPTW est un sous-problème important au sein d'une large classe de problèmes dits de tournées de véhicules. Il correspond également à un problème d'ordonnancement sur une seule machine avec un temps de set-up dépendant de la séquence et des dates de disponibilités et de fins des tâches. Le TSPTW est évidemment au moins aussi difficile que le TSP et c'est donc un problème NP-Difficile [21]. Dans cette étude, nous étudions le TSPTW avec une approche Programmation Par Contraintes (CP). Nous évaluons différents modèles réalisant une propagation plus ou moins forte. Nous nous intéresserons, par ailleurs, à différentes stratégies de recherche et examinons le meilleur compro-

mis entre les modèles et les stratégies de recherche. Cette étude est utile pour identifier le couple modèle/stratégie le plus robuste afin de l'inclure dans un solveur de problèmes de tournées de véhicules.
Après une brève introduction sur les travaux déjà effectués (section 2), nous proposons d'abord trois modèles en CP (section 4). Nous examinons ensuite des techniques de propagation afin d'améliorer le filtrage (section 5). Enfin, différentes stratégies de recherche sont abordées (section 6). Dans la dernière section (section 7), les résultats seront présentés et comparés.

2 État de l'art

L'une des premières approches exactes pour résoudre le TSPTW est proposée par [8]. Elle est basée sur une procédure de Branch & Bound dont la borne inférieure est obtenue à l'aide de la programmation dynamique. Une approche par Branch & Bound est également présentée dans [2] avec une borne inférieure s'appuyant sur le dual de la formulation proposée. Ces approches minimisent le temps total et non le temps de trajet qui a été introduit dans [16] avec un programme linéaire où l'on retrouve une formulation à base de flots. Des contraintes de précédences sont aussi utilisées et couplées avec une méthode de réduction des fenêtres de temps proposée dans [10]. Cette méthode réduit les bornes des fenêtres de temps en examinant les successeurs et prédécesseurs ainsi que la contrainte de temps. Dans [1] et plus récemment dans [9], les auteurs résolvent la version asymétrique du TSPTW par une méthode de Branch-and-Cut. Plusieurs techniques de propagation sont aussi proposées, des pré-traitements sur les données ou encore des heuristiques pour obtenir une première solution. À ce jour l'une des meilleures approches exactes s'appuie sur une programmation dynamique utilisant des bornes inférieures obtenues par génération de colonnes [3].
En Programmation Par Contraintes, deux approches ont été étudiées. La première est celle de [20] dans laquelle les auteurs proposent des contraintes redondantes permettant de réduire l'espace de recherche. En particulier, ils utilisent la réduction des fenêtres de temps couplée avec une contrainte d'élimination des arcs. Cette dernière met à jour systématiquement deux ensembles pour chaque noeud représentant les noeuds pouvant se placer avant ou après le noeud considéré. Les auteurs utilisent aussi une stratégie de recherche efficace qui est basée sur la fréquence d'apparition d'un client dans le domaine des successeurs ou prédécesseurs. Dans la seconde approche [12], les auteurs se basent sur le modèle de [20] et ajoutent des bornes inférieures utilisant un problème d'affectation. Ils utilisent cette borne inférieure pour réaliser un filtrage par les coûts et l'améliorent avec une méthode de plans coupants.
Des heuristiques\metaheuristiques ont aussi été proposées afin d'obtenir des résultats rapidement sans avoir de preuve d'optimalité. Dans [13], une heuristique d'insertion est proposée. On peut également citer [6] qui utilisent un algorithme de recuit simulé, version améliorée dans [19]. Plus récemment, un algorithme Beam-ACO [5, 18] est développé pour le TSPTW. De plus, les auteurs proposent une formalisation des instances de TSPTW dans un format standard.

3 Problème et notations

On considère un graphe orienté complet $G(N, E)$ dans lequel E est l'ensemble des arcs et $N = \{0, \ldots, n+1\}$ est l'ensemble des noeuds représentant les clients et le dépôt. On distingue de plus les ensembles suivants $N^s = \{0, \ldots, n\}$ et $N^e = \{1, \ldots, n+1\}$). Les noeuds 0 et $n+1$ correspondent au dépôt : le noeud 0 est le dépôt d'origine où la route commence, le noeud $n+1$ est le dépôt de destination où la route se termine. Nous associons une fenêtre de temps $[a_i, b_i]$ pour chaque noeud $i \in N^e$. On considère que le voyageur part à l'instant $t = 0$ du noeud 0 et que la fenêtre du noeud $n+1$ est donc du type $[0, b_{n+1}]$ indiquant que le chemin doit être complété avant l'heure b_{n+1}. Une distance $d_{ij} \in \mathbb{N}$ et un temps $t_{ij} \in \mathbb{N}$ de trajet sont associés à chaque arc (i, j). Le temps de service du client i est inclus, sans perte de généralité, dans chaque temps de trajet t_{ij} vers les noeuds $j \in N - \{i\}$.

4 Modèles

Par la suite $\overline{x}$ (resp. $\underline{x}$) désigne la borne supérieure (resp. la borne inférieure) de la variable x et $D(x)$ indique son domaine (ensemble fini de valeurs possibles pour x).

4.1 Premier modèle

Le modèle de base en Programmation Par Contraintes s'appuie sur les variables $next_i$ pour chaque noeud i. $next_i \in [1, \ldots, n+1]$ représente le successeur immédiat du noeud i dans la tournée (par convention $next_{n+1} = 0$). De façon symétrique, la variable $pred_i \in [0, \ldots, n]$ représente le prédécesseur du client i dans la tournée ($pred_0 = n+1$). Des variables représentant une accumulation de quantité complètent le modèle : $dist$ et $start$. $dist_i \in [0, \ldots, M]$, représente pour chaque noeud i la distance parcourue en arrivant au noeud i tandis que $start_i \in [a_i, \ldots, b_i]$, représente le temps (ou l'heure) d'arrivée chez le client i. M représente ici une borne supérieure de la distance totale

de parcours.

Le modèle de base (M_1) est le suivant :

$$\textbf{Minimize } z \tag{1}$$

$$z = \sum_{i=0}^{n}(d_{i,next_i}) = \sum_{i=1}^{n+1}(d_{pred_i,i}) \tag{2}$$

$$\textsc{Circuit}(next_0, ..., next_{n+1}) \tag{3}$$

$$(M_1) \quad dist_{next_i} = dist_i + d_{i,next_i} \qquad \forall i \in N^s \tag{4}$$

$$dist_0 = 0 \tag{5}$$

$$start_{next_i} \geq start_i + t_{i,next_i} \qquad \forall i \in N^s \tag{6}$$

$$start_0 = 0 \tag{7}$$

La contrainte (3), présentée dans [17], permet de maintenir un circuit dans le graphe en visitant une seule fois chaque noeud. Cette contrainte implique notamment $\textsc{AllDifferent}(next_0, \ldots, next_{n+1})$. Les contraintes (4) et (6) s'appuient sur l'utilisation de la contrainte Element [24] afin de maintenir l'accumulation de quantités le long de la route. Les variables *pred* sont liées aux variables *next* par la propriété suivante : $next_i = j \Leftrightarrow pred_j = i$ pour chaque paire (i, j). Cette propriété se traduit en terme de Programmation Par Contraintes par l'utilisation de la contrainte globale Inverse :

$$\textsc{Inverse}([next_0, \ldots, next_{n+1}], [pred_0, \ldots, pred_{n+1}]) \tag{8}$$

L'utilisation des prédecesseurs renforce la borne inférieure sur z car les quantités $\sum_{i=0}^{n}(\underline{d_{i,next_i}})$ et $\sum_{i=1}^{n+1}(\underline{d_{pred_i,i}})$ ne sont pas forcément égales ce qui explique les contraintes d'égalité (2). Cependant ces quantités sont égales lorsque les variables sont instanciées. De plus ces variables offrent des possibilités supplémentaires pour le branchement.

4.2 Contraintes redondantes

En CP, l'utilisation de contraintes redondantes permet le renforcement de la propagation. La contrainte sur l'élimination des arcs proposée dans [16], permet l'élimination de valeurs dans le domaine des *next* en considérant les fenêtres de temps.

$$start_i + t_{ij} > start_j \Rightarrow next_i \neq j \quad \forall (i, j) \in N^s \times N^e \; i \neq j \tag{9}$$

Cette dernière contrainte est couplée avec la contrainte de réduction des fenêtres de temps présentée dans [10]. En effet, les domaines des successeurs et prédécesseurs peuvent être utilisés pour réduire les fenêtres de temps.

$$start_i \geq \min_{k \in D(pred_i)}(start_k + t_{ki}) \qquad \forall i \in N^e \tag{10}$$

$$start_i \leq \max_{k \in D(next_i)}(start_k - t_{ik}) \qquad \forall i \in N^s \tag{11}$$

On notera que la contrainte (6) n'est pas équivalente en termes de propagation. Les dernières contraintes présentées sont utilisées dans [20]. Ces raisonnements peuvent être renforcés en remplaçant les temps de trajets t_{ab} par le temps du plus court chemin entre a et b et en considérant non plus les successeurs et prédécesseurs directs mais les clients devant être visités après ou avant. Toutefois, dans [12], les expérimentations ont montré que le calcul du plus court chemin était trop lourd pour le gain obtenu.

Nous présentons maintenant deux modèles permettant d'améliorer le modèle de base.

4.3 Modèle booléen

On ajoute à (M_1) des variables booléennes b_{ij} pour chaque couple (i, j). Ces dernières permettent d'identifier l'ordre relatif d'un client i par rapport à un client j : $b_{ij} = 1$ si le client i est visité avant le client j et 0 sinon. Ainsi au lieu de raisonner uniquement sur les successeurs/prédécesseurs directs, on s'intéresse aux clients situés après/avant dans le chemin. Les contraintes associées à ce modèle sont les suivantes $\forall (i, k, j) \in N^3$ s.t $i \neq k \neq j$:

$$b_{ik} + b_{ki} = 1 \tag{12}$$

$$(b_{ik} = 1 \wedge b_{kj} = 1) \Rightarrow b_{ij} = 1 \tag{13}$$

$$(M_2) \quad (b_{ik} = 1) \Rightarrow start_k \geq start_i + t_{ik} \tag{14}$$

$$(b_{ik} = 1) \wedge (b_{kj} = 1) \Rightarrow next_i \neq j \tag{15}$$

$$(b_{ik} = 1) \Rightarrow next_k \neq i \tag{16}$$

Nous soulignons ici la contrainte (15) dont l'apport peut être important (voir aussi [20]). L'inconvénient de ce modèle est que le nombre de contraintes est en $\mathcal{O}(n^3)$. Ces raisonnements peuvent constituer l'objet d'une contrainte globale manipulant l'ensemble du graphe de précédences pour éviter un problème de mémoire. Cette contrainte peut en particulier assurer la fermeture transitive faite par (13) et la propagation vers les variables *next* faite par (15). Nous examinons un modèle intermédiaire dans la section suivante.

4.4 Modèle position

Nous conservons l'esprit général du modèle booléen, mises à part les contraintes (13) et (15) qui sont trop nombreuses. Les variables b_{ij} sont toujours présentes mais nous ajoutons des variables pos_i représentant la position du noeud i dans la tournée :

$$pos_i = \sum_{j \in N \setminus \{i\}} b_{ji} \qquad \forall i \in N \tag{17}$$

Les contraintes de ce modèle sont les suivantes, $\forall (i,k) \in N^2$ s.t $i \neq k$:

$$b_{ik} + b_{ki} = 1 \quad (18)$$
$$(b_{ik} = 1) \Rightarrow start_k \geq start_i + t_{ik} \quad (19)$$
$$(M_3) \quad (b_{ik} = 1) \Rightarrow next_k \neq i \quad (20)$$
$$pos_k > pos_i + 1 \Rightarrow next_i \neq k \quad (21)$$
$$pos_k > pos_i \Leftrightarrow b_{ik} = 1 \quad (22)$$
$$\text{AllDifferent}(pos_0, \ldots, pos_{n+1}) \quad (23)$$

Ainsi la contrainte (15) est remplacée par (22). On notera que le filtrage obtenue par (22) est plus faible que celui de (15).

Ce modèle est adapté à la mise en oeuvre de raisonnements énergétiques semblables à ceux de la contrainte Disjunctive. Notre expérience est que l'application du filtrage de la Disjunctive est trop coûteux dans ce contexte et pas toujours utile. Nous proposons ici un raisonnement qui nous semble un bon compromis entre temps et filtrage. Il s'agit de mettre à jour une variable de position pos_i par rapport à l'ensemble des autres clients et de leurs possibilités de placement avant ou après le client i.

Soit A_i l'ensemble des visites qui peuvent être placées après le client i et B_i l'ensemble des clients pouvant être avant ou après le client i :

$$A_i = \{j \in N \mid \underline{b_{ij}} = 1\}$$
$$B_i = \{j \in N \mid \overline{b_{ij}} = 1 \wedge \underline{b_{ij}} = 0\}$$

Le temps minimum requis après i est noté EA_i :

$$EA_i = \underline{t_{i,next_i}} + \sum_{j \in A_i} \underline{t_{j,next_j}}$$

Dans un premier temps, on note que la borne supérieure de $start_i$ ne peut pas dépasser le temps minimum requis par les visites de A_i :

$$\overline{start_i} \leq \overline{start_{n+1}} - EA_i$$

Dans un second temps, nous filtrons la borne inférieure de pos_i. Considérons les visites $x_1, \ldots, x_k$ de B_i rangées par ordre croissant de distances aux successeurs :

$$\underline{t_{x_1,next_{x_1}}} \leq \underline{t_{x_2,next_{x_2}}} \leq \underline{t_{x_3,next_{x_3}}} \leq \ldots \leq \underline{t_{x_k,next_{x_k}}}$$

Soit c le plus grand entier tel que :

$$EA_i + \sum_{j=0}^{c} \underline{t_{x_j,next_{x_j}}} \leq (\overline{start_{n+1}} - \underline{start_i})$$

$c + |A_i|$ est donc le nombre de visites maximum qui peuvent avoir lieu après la visite i, ce qui nous donne une borne inférieure pour pos_i :

$$\underline{pos_i} \geq n - (c + |A_i| + 1)$$

Un raisonnement symétrique est effectué pour la borne inférieure de $start_i$ et la borne supérieure de pos_i en prenant en compte le nombre minimum de visites pouvant se placer avant le client i.

Ce raisonnement est un cas particulier de raisonnement énergétique que ferait la contrainte Disjunctive [7] par l'algorithme d'*edge-finding*. Comme mentionné en introduction, le TSPTW peut se voir comme un problème d'ordonnancement sur une machine avec un temps de setup dépendant de la séquence. On peut écrire un modèle CP s'appuyant sur une Disjunctive avec des durées variables pour les tâches. En ajoutant une variable end_i telle que $start_i + t_{i,next_i} = end_i$. Ce modèle n'est pas présenté plus en détails car sa mise en oeuvre n'a pas été concluante pour le moment.

En ignorant le raisonnement énergétique proposé sur les positions, on peut ranger les modèles selon le niveau de propagation atteint. En effet, le modèle de base ($M1$) possède un niveau de propagation plus faible comparé au modèle booléen ($M2$) qui possède la plus forte propagation. Le modèle position ($M3$) est un modèle intermédiaire en terme de propagation.

5 Amélioration de la propagation

Les modèles précédents souffrent de l'absence d'une borne inférieure globale de la fonction objectif. Nous proposons donc de remplacer la contrainte Circuit par la contrainte WeightedCircuit présentée dans [4]. Cependant, les techniques coûteuses de filtrage orientées par le coût se rentabilisent d'autant plus que la borne supérieure (en cas de minimisation) est une borne de qualité au noeud racine. En effet, tous les raisonnements s'appuient sur l'écart à la meilleure solution connue au moment de leur application. Nous proposons donc de calculer une borne supérieure de l'objectif avant la recherche par une heuristique simple.

5.1 Borne supérieure

Pour obtenir une borne supérieure au noeud racine, nous effectuons une recherche locale très simple. Deux opérateurs de voisinage sont appliqués sur une séquence (initialement aléatoire) de clients : l'échange de deux clients ou le déplacement d'un client dans la séquence. Le premier mouvement améliorant trouvé est effectué. Un mouvement est améliorant s'il réduit le nombre de clients hors de leur fenêtre de temps ou s'il réduit la distance parcourue sans augmenter le nombre de clients visités à l'extérieur de leur fenêtre (optimisation lexicographique). L'algorithme termine en atteignant un minimum local, c'est à dire une séquence

de clients non améliorable par ces deux opérateurs. K exécutions de cet algorithme sont effectuées à partir de séquences aléatoires et la meilleure solution réalisable est utilisée pour initialiser la borne supérieure de l'objectif.

5.2 Weighted Circuit

La seconde technique employée est le remplacement de la contrainte CIRCUIT par la contrainte WEIGHTEDCIRCUIT de [4]. Cette contrainte propage une borne inférieure de z et filtre des valeurs des variables $next$ du point de vue du coût. La WEIGHTEDCIRCUIT permet de maintenir un circuit dans un graphe pondéré tout en considérant la minimisation du poids total. Elle correspond au problème du Voyageur du Commerce :

$$\text{WEIGHTEDCIRCUIT}([next_0, \ldots, next_{n+1}], z)$$

L'algorithme de filtrage est basé sur la relaxation du TSP qui utilise la "1-tree relaxation" introduite par Held et Karp [14]. La "1-tree relaxation" relâche la contrainte de degré du TSP et utilise la structure du 1-tree. Cette dernière correspond à un arbre dans le graphe contenant les noeuds $\{1, \ldots, n+1\}$ et à deux arcs incidents au noeud 0. Nous cherchons, dans cette relaxation, un 1-tree de poids minimum. Un tour dans le TSP est un cas particulier de 1-tree. La borne de Held et Karp s'obtient par relaxation Lagrangienne [14, 15] (pour une étude de la borne de Held et Karp, se référer à [23]).
La relaxation lagrangienne est basée sur le potentiel de chaque noeud et le coût lié au potentiel pour chaque arc. Le potentiel d'un noeud représente le coût de violation de la contrainte de degré. Ainsi, plus le potentiel d'un noeud est élevé, plus le coût d'un arc incident sera élevé. Par conséquent, à chaque itération l'algorithme aura tendance à converger vers un 1-tree de poids minimum respectant la contrainte de degré. Lors de l'algorithme de filtrage, nous pouvons distinguer deux phases : l'identification des arcs interdits et l'identification des arcs obligatoires.

Identification des arcs interdits Le calcul du coût marginal d'un arc e est nécessaire pour l'identification des arcs interdits. Ce coût représente l'augmentation du coût du 1-tree si e devait être un arc du 1-tree de poids minimum. Ainsi, si le coût global du 1-tree de poids minimum plus le coût marginal d'un arc e est plus grand que la borne supérieure de z, alors e est un arc interdit et donc il faut le supprimer du graphe.

Identification des arcs obligatoires. L'identification des arcs obligatoires est similaire à l'identification des arcs interdits. Le coût de remplacement d'un arc e représente le coût supplémentaire du 1-tree de poids minimum lorsque l'arc e est remplacé par un arc ne faisant pas partie du 1-tree. Si le coût de remplacement d'un arc e associé au coût du 1-tree est supérieur à la borne supérieure de z, alors l'arc e est un arc obligatoire. Cependant, nous ne pouvons pas encore utiliser ce résultat car notre problème de base est un problème dans un graphe orienté. Nous devons pour cela, vérifier le domaine des variables $next$ associé à l'arc e afin d'identifier si une arête est obligatoire.

6 Stratégies de recherche

Dans cette section, nous abordons différentes stratégies de recherche pour les modèles présentés précédemment. Nous examinons des schémas de branchement consistant à affecter une variable à une valeur de son domaine.

Plusieurs stratégies applicables à tous les modèles sont présentées ainsi que deux stratégies spécifiques aux modèles booléen (M_2) et position (M_3).

6.1 Stratégies de recherche génériques

PathMaintain : Cette stratégie consiste à étendre un chemin partant du noeud 0. Après avoir affecté $next_i$ à j, on sélectionne donc la variable $next_j$ pour prolonger le chemin. La valeur choisie est le noeud ayant sa fenêtre de temps la plus proche.

MinDomNextPred : Ce schéma de branchement est souvent utilisé en CP. La variable de plus petit domaine parmi toutes les variables $next$ et $pred$ est choisie en priorité. La valeur choisie est la plus petite valeur dans le domaine de la variable sélectionnée.

HeuristiquePesant : L'heuristique de Pesant est le schéma de branchement utilisé dans [20].

1. Soit s la taille du plus petit domaine des $next$ et $pred$.
   Soit $\mathcal{V} = \{next_i | \ |D(next_i)| = s, \forall i = 1, \ldots, n\} \cup \{pred_i | \ |D(pred_i)| = s, \forall i = 1, \ldots, n\}$.
2. Si $|\mathcal{V}| = 1$, choisir la variable contenue dans $\mathcal{V}$
3. Sinon
   (a) Pour chaque élément e dans $\cup_{v \in \mathcal{V}} D(v)$, calculer $e^{\#}$ le nombre d'apparition de e dans le domaine des variables de $\mathcal{V}$.
   (b) Choisir la variable qui maximise $f(v) = \sum_{e \in D(v)} e^{\#}$.

Nous affectons par la suite la valeur correspondant au client le plus proche (en distance).

NoGoodsRecording : Un nogood est *une affectation partielle qui ne peut pas être étendue à une solution faisable*. Ainsi, toute affectation contenant un nogood est soit irréalisable du point de vue des fenêtres de temps, soit de coût supérieur à $\overline{z}$. L'enregistrement de nogoods peut permettre d'éviter l'exploration redondante de sous-arbres de recherche. L'approche proposée ici impose un schéma de branchement s'appuyant sur PathMaintain. Une affectation des variables *next* depuis le noeud dépôt constitue donc un chemin partiel. Cette affectation est définie par l'ensemble $S \subset N$ de clients visités sur ce chemin, le **dernier** client noté k ($k \in S$), la distance d parcourue jusqu'au dernier client et le temps t indiquant le service au plus tôt du client k. On caractérise donc une affectation partielle de l'heuristique PathMaintain par le quadruplet (S, k, d, t). Si l'affectation (S_1, k_1, d_1, t_1) est un nogood, alors il est inutile d'essayer d'étendre toute affectation (S_2, k_2, d_2, t_2) telle que :

$$S_2 = S_1,\ \ k_2 = k_1,\ \ d_2 \geq d_1 \ \ \text{et} \ \ t_2 \geq t_1$$

Sachant que (S_1, k_1, d_1, t_1) est un nogood, il est inutile d'explorer un chemin partiel contenant les mêmes clients S (pas forcément dans le même ordre), terminant au même noeud k_1 et arrivant plus tard en k_1 tout en ayant parcouru une distance plus longue.
On enregistre les nogoods au backtrack pendant le déroulement de l'heuristique PathMaintain. De plus, à partir du dernier client i sur le chemin partiel, le client $j \in D(next_i)$ est éliminé du domaine de $next_i$ si le chemin partiel obtenu est prouvé irréalisable par un nogood connu.
L'heuristique de branchement PathMaintain permet de se placer dans le même espace de recherche que l'approche Programmation Dynamique proposée dans [11]. L'enregistrement des nogoods permet de couper certaines branches de l'espace de recherche cité précédemment.

6.2 Stratégie de recherche pour le modèle booléen

Dans cette section, nous présentons une stratégie de recherche dédiée au modèle booléen (section 4.3).

ImpactBoolVar : Cette stratégie de recherche consiste à mesurer l'impact de branchement des b_{ij} sur la réduction des fenêtres de temps. Cet impact correspond à la somme des réductions potentielles sur les domaines des variables *start* lorsque la variable est égale à 0 ou 1. La variable provoquant la plus forte réduction est choisie en priorité.

1. Pour chaque paire de noeud (i, j), les réductions de domaine sont notées $startGain(i, j)$, $endGain(i, j)$, $startGain(j, i)$, $endGain(j, i)$ selon que $b_{ij} = 1$ ou $b_{ij} = 0$
   — $startGain(i, j) = \max(0, \underline{start_i} + t_{ij} - \underline{start_j})$
   — $endGain(i, j) = \max(0, \overline{start_i} - (\overline{start_j} - t_{ij}))$
2. Sélectionner la variable b_{ij} qui maximise $f(i, j) = startGain(i, j) + endGain(i, j) + startGain(j, i) + endGain(j, i)$
3. Appliquer $b_{ij} = 1$

6.3 Stratégie de recherche pour le modèle position

Cette section présente une stratégie de recherche dédiée au modèle position (section 4.4).

InflectionPoint : Le principe de cette heuristique est d'appliquer l'heuristique MinDomNextPred sur des groupes de variables identifiés au noeud racine après la propagation initiale. Ces groupes constituent des ensembles de positions indépendants. En effet, on remarque qu'une position instanciée au noeud racine sépare le problème en deux ensembles de positions disjoints.

7 Résultats

Les résultats sont séparés en deux parties. La première partie est l'analyse des techniques de renforcement de la propagation (section 5). La seconde partie est l'analyse des couples modèle/stratégie de recherche. Cette analyse est une première base pour le développement d'une relaxation prenant en compte les fenêtres de temps ainsi que les distances (coûts). Le développement est fait en c++ avec la librairie or-tools [25] pour la Programmation Par Contraintes. Les tests ont été effectués sur un Intel Xeon 4 coeurs 2.27 GHz avec 8.00 Go de mémoire avec une limite de temps de 1200 secondes.

7.1 Amélioration de la propagation

Le but de cette section est de montrer l'apport des différentes techniques utilisées afin d'améliorer la propagation (section 5). Les instances utilisées sont les neufs premières instances de Pesant *et al.* [20]. Ce sont des instances dérivées des problèmes RC2 de Solomon [22] pour le problème VRPTW. Le modèle utilisé est le modèle booléen et la stratégie de recherche est MinDomNextPred. Toutes les instances sont résolues à l'optimum. Deux critères d'évaluation sont utilisés, le temps de résolution en seconde et le nombre d'échecs dans l'arbre de recherche.

La Figure 1 présente les résultats obtenus sur les neufs instances pour les différentes techniques. Basis représente le modèle de base sans ajouts. UpperBound est l'ajout de la borne supérieure sur le modèle de base. Nous pouvons voir que cet ajout apporte déjà beaucoup pour filtrage. WeightedCircuit est le remplacement de Circuit par WeightedCircuit. Même

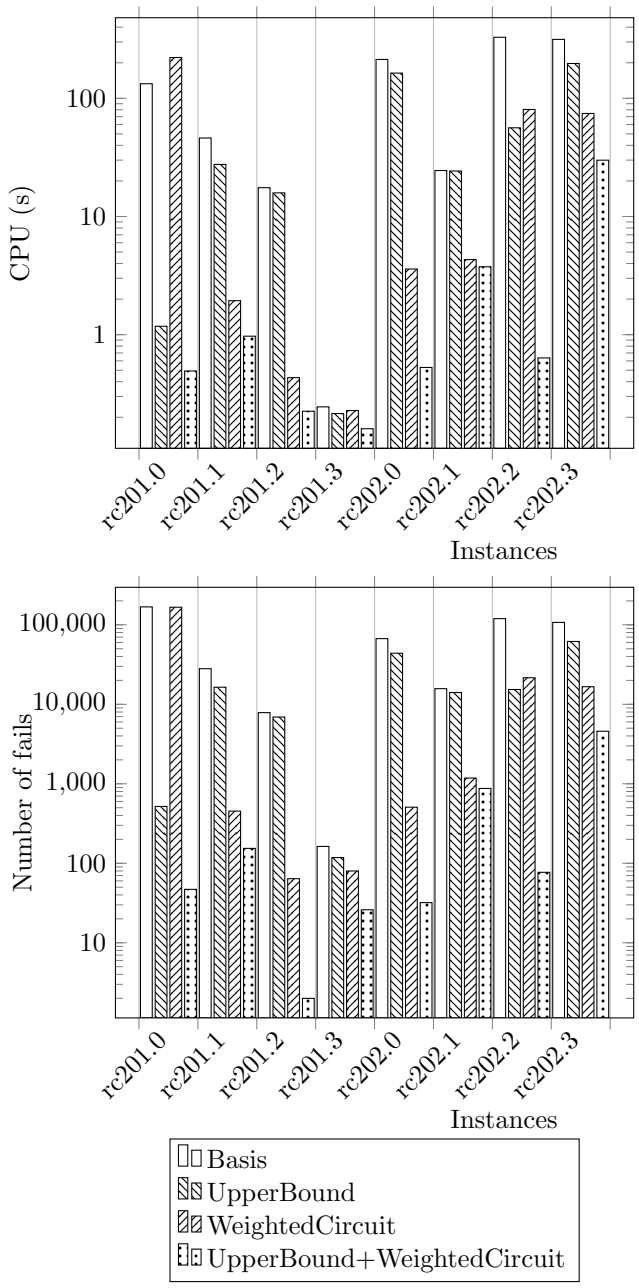


FIGURE 1 – Amélioration de la propagation

si l'algorithme de filtrage est lourd (voir rc201.0), le gain reste significatif. Cependant, l'ajout des deux techniques UpperBound+WeightedCircuit est encore plus performant et permet de réduire significativement l'arbre de recherche.

7.2 Résultats des différents modèles et stratégies de recherche

Dans cette section, nous présentons les différentes performances des couples modèle/stratégie de recherche. Dans un premier temps, nous verrons le gain apporté par les NogoodsRecording sur la stratégie de recherche PathMaintain. Dans un second temps, l'ensemble des couples sera étudié. Au vu des résultats de la section précédente, nous décidons de rajouter les techniques de propagation à tous les modèles (base, booléen, position). Les instances utilisées sont les instances 20 et 40 clients de Dumas [11]. Chaque instance est divisée en groupe de 5 problèmes (w20, w40, w60, w80, w100). La différence entre les groupes est la largeur des fenêtres de temps. Pour les problèmes w20, les fenêtres de temps sont très petites entrainant une propagation importante sur l'ordre des visites. En revanche, avec les problèmes w100, les fenêtres de temps sont beaucoup plus larges.

PathMaintain/NoGoods : La Figure 2 présente les différents modèles (base, booléen et position) couplés aux stratégies de recherche PathMaintain et NogoodsRecording sur les instances à 20 clients.

Nous nous apercevons que le modèle de base n'est pas compétitif par rapport aux deux autres modèles. Au delà de w60, le modèle de base n'arrive pas à résoudre les problèmes. Évidement, le modèle position filtre beaucoup moins que le modèle booléen (le nombre d'échecs pour le modèle position est supérieure au modèle booléen). Cependant, pour des problèmes avec des fenêtres de temps petites (w20, w40) le modèle position est plus rapide que le modèle booléen. De plus, le schéma de branchement NogoodsRecording entraîne une diminution du nombre d'échecs par rapport à la stratégie de recherche PathMaintain. Cependant, cette dernière n'est pas aussi performante que d'autres schémas de branchement.

Couple modèle/stratégie de recherche : La Figure 3 montre les différents couples modèle/stratégie de recherche sur les instances 40 clients pour des petites fenêtres de temps (w20, w40) et les instances 20 clients pour le reste.

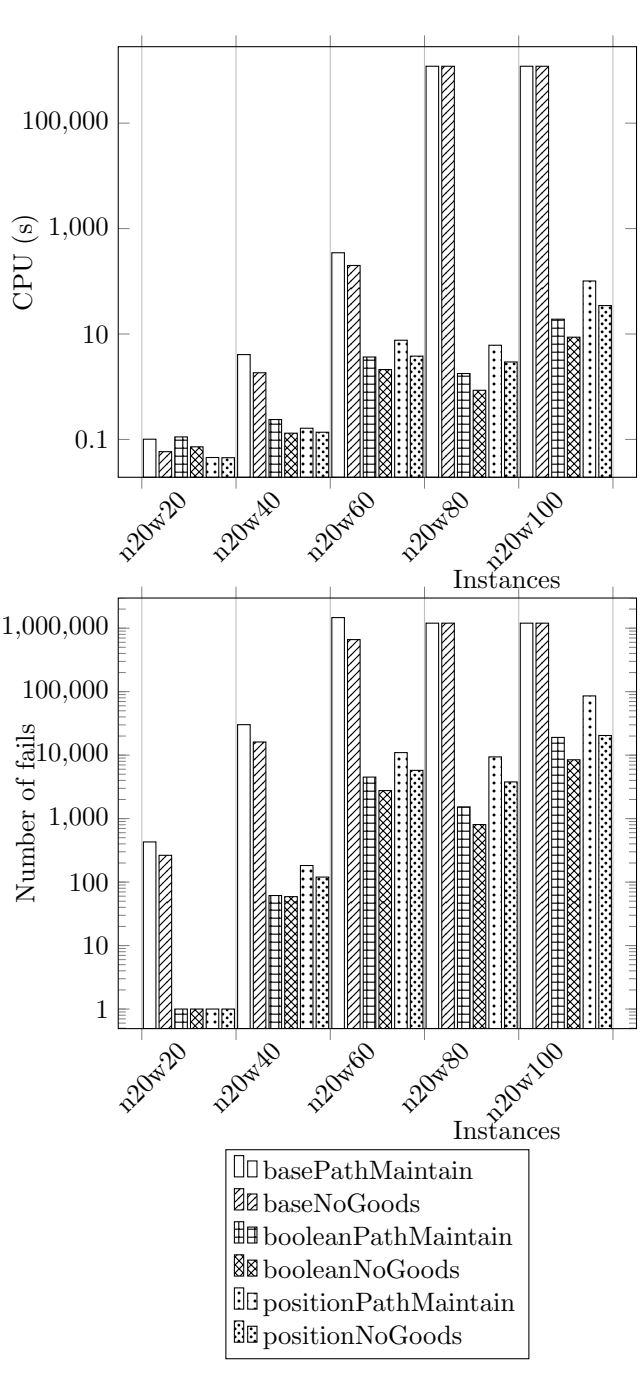


FIGURE 2 – Différence entre modèles/PathMaintain ou NoGoodsRecording

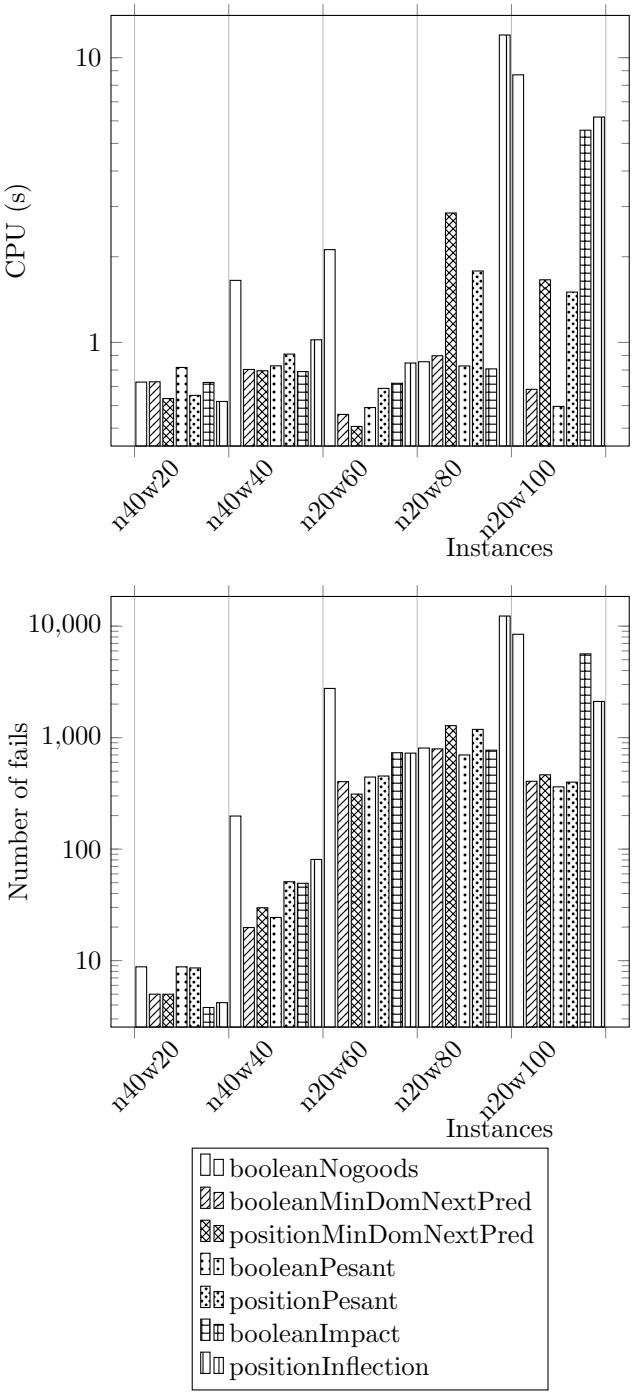


FIGURE 3 – Différence entre modèles/stratégies de recherche

Instances	Résultats	position/HeuriticPesant		
	[20], [12]	Solution	Temps(s)	# d'échec
rc201.0	628.62^{p}	628.62	0.59	97
rc201.1	654.70^{p}	654.7	0.70	150
rc201.2	707.65^{p}	707.65	0.17	2
rc201.3	422.54^{p}	422.54	0.10	23
rc202.0	496.22^{p}	496.22	0.56	50
rc202.1	426.53^{p}	426.53	2.95	950
rc202.2	611.77^{p}	611.77	5.60	958
rc202.3	627.85^{p}	627.85	38.72	7573
rc203.0	.	.	1200	198002
rc203.1	.	.	1200	104930
rc203.2	.	617.46	148.8	24266
rc204.0	.	541.45	1.47	79
rc204.1	.	485.37	0.49	38
rc204.2	.	.	1200	74351
rc205.0	511.65^{p}	511.65	4.90	1452
rc205.1	491.22^{p}	491.22	0.16	14
rc205.2	714.69^{f}	715.34*	1200	418318
rc205.3	601.24^{p}	601.24	38.14	10909
rc206.0	.	835.23*	1200	129235
rc206.1	.	664.73	2.06	254
rc206.2	655.37^{p}	655.37	6.42	1219
rc207.0	806.69^{f}	806.69	953.93	37390
rc207.1	726.36^{f}	726.36	8.60	521
rc207.2	546.41^{p}	546.41	0.90	55
rc208.0	.	.	1200	48194
rc208.1	.	509.04	7.77	914
rc208.2	.	503.92	0.38	4

^{p} : Optimal value from [20], ^{f} : Optimal value from [12],* : Optimality not proved

Tableau 1 – Résultats des instances proposées par Pesant *et al.* [20] avec le modèle position/heuristique de Pesant

Tout d'abord, nous nous apercevons que les stratégies de recherche dédiées à un modèle ne sont pas performantes par rapport à l'heuristique Pesant *et al.* ou MinDomNextPred. Ensuite, le modèle position est efficace, en terme de temps de résolution, pour les problèmes avec des petites fenêtres de temps (w20, w40, w60). Cependant, lorsque les fenêtres de temps deviennent larges, la vision globale apportée par le modèle booléen (modèle avec le plus fort raisonnement de propagation) est plus performante. Enfin, la stratégie de recherche des NogoodsRecording ne semble pas prometteuse.

Comparaison avec les résultats de la littérature : Le Tableau 1 propose une comparaison des résultats avec les différentes études en Programmation Par Contraintes dans [20] et [12] sur les instances de Solomon et Pesant *et al.*.

Nous résolvons à l'optimum sept instances de plus que les approches proposées dans [20] et [12]. Toutefois, l'optimum de l'instance rc205.2 n'est pas atteint alors que [12] prouve l'optimalité. Les instances non résolues sont les instances avec un nombre de clients supérieur à 35. Notons que la totalité de ces benchmarks a été résolue par [3].

8 Conclusion

Nous avons tout d'abord étudié trois modèles en Programmation Par Contraintes pour le TSPTW possédant des niveaux de propagation différents. Nous avons ensuite constaté que ces modèles étaient fortement renforcés en remplaçant la contrainte Circuit par WeightedCircuit couplée au calcul d'une borne supérieure. Enfin, différentes stratégies de recherche ont été testées. Cependant, on s'aperçoit que les stratégies de recherche efficaces sont celles déjà largement utilisées dans la littérature (HeuristiquePesant, MinDomNextPred). Cette étude constitue un travail préliminaire pour évaluer l'intérêt d'un filtrage global orienté par les coûts pour le TSPTW. Notre objectif est de mettre en oeuvre ce filtrage sur des problèmes plus généraux de tournées de véhicules.

Références

[1] Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. Solving the asymmetric travelling salesman problem with time windows by Branch-and-Cut. *Mathematical Programming*, 90(3) :475–506, 2001.

[2] Edward K. Baker. Technical note—an exact algorithm for the time-constrained traveling salesman problem. *Operations Research*, 31(5) :938–945, 1983.

[3] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(3) :356–371, 2012.

[4] Pascal Benchimol, Willem-Jan Van Hoeve, Jean-Charles Régin, Louis-Martin Rousseau, and Michel Rueher. Improved filtering for weighted circuit constraints. *Constraints*, 17(3) :205–233, 2012.

[5] Christian Blum. Beam-ACO for simple assembly line balancing. *INFORMS Journal on Computing*, 20(4) :618–627, 2008.

[6] Stephen P. Brooks and Byron J.T. Morgan. Optimization using simulated annealing. *The Statistician*, pages 241–257, 1995.

[7] Jacques Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1) :42 – 47, 1982.

[8] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2) :145–164, 1981.

[9] Sanjeeb Dash, Oktay Günlük, Andrea Lodi, and Andrea Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1) :132–147, 2012.

[10] Martin Desrochers, Jacques Desrosiers, and Marius M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2) :342–354, 1992.

[11] Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2) :367–371, 1995.

[12] Filippo Focacci, Andrea Lodi, and Michela Milano. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing*, 14(4) :403–417, 2002.

[13] Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3) :330–335, 1998.

[14] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6) :1138–1162, 1970.

[15] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees : Part II. *Mathematical Programming*, 1(1) :6–25, 1971.

[16] André Langevin, Martin Desrochers, Jacques Desrosiers, Sylvie Gélinas, and François Soumis. A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Networks*, 23(7) :631–640, 1993.

[17] Jean-Louis Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1) :29–127, 1978.

[18] Manuel López-Ibáñez and Christian Blum. Beam-ACO for the travelling salesman problem with time windows. *Computers & Operations Research*, 37(9) :1570–1583, 2010.

[19] Jeffrey W. Ohlmann and Barrett W. Thomas. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 19(1) :80–90, 2007.

[20] Gilles Pesant, Michel Gendreau, Jean-Yves Potvin, and Jean-Marc Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32(1) :12–29, 1998.

[21] Martin W.P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1) :285–305, 1985.

[22] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2) :254–265, 1987.

[23] Christine L. Valenzuela and Antonia J. Jones. Estimating the Held-Karp lower bound for the geometric TSP. *European Journal of Operational Research*, 102(1) :157–175, 1997.

[24] Pascal Van Hentenryck and Jean-Philippe Carillon. Generality versus specificity : An experience with AI and OR techniques. In *AAAI*, pages 660–664, 1988.

[25] Nikolaj van Omme, Laurent Perron, and Vincent Furnon. or-tools user's manual. Technical report, Google, 2014.

Autour de la décomposition des contraintes non-binaires en contraintes binaires équivalentes *

Achref El Mouelhi

Aix-Marseille Université, LSIS UMR 7296
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
achref.elmouelhi@lsis.org

Résumé

Des efforts considérables en recherche ont été déployés pour la conversion d'une instance CSP non-binaire en une instance CSP binaire équivalente. Ces travaux peuvent être subdivisés en deux. Les premiers ont été consacrés à l'étude du codage binaire des instances non-binaires. Trois codages ont été proposés à savoir le codage dual, le codage par variables cachées et le codage double. Malheureusement, ces codages ne permettent pas d'utiliser des propriétés et des résultats intéressants restrictifs au cas binaire. Les deuxièmes consistent à transformer chaque contrainte non-binaire en un ensemble de contraintes binaires, l'instance obtenue est appelée primale. Malheureusement, cette transformation ne préserve pas la satisfiabilité.

Dans cet article, nous proposons deux conditions dont la présence garantit de pouvoir remplacer une contrainte non-binaire en un ensemble de contraintes binaires, tout en préservant la satisfiabilité. Une étude expérimentale prouve que notre approche n'est pas artificielle puisque certains benchmarks ternaires peuvent être transformés en instances binaires équivalentes et par la suite être efficacement résolues par des algorithmes de l'état de l'art comme MAC.

Abstract

Considerable research efforts have been focused on the translation of non-binary CSP into an equivalent binary CSP. Most of this work was devoted to studying the binary encoding of non-binary CSP. Three encodings have been proposed namely dual encoding, hidden variable encoding and double encoding. Unfortunately, such encodings do not allow to use some properties and interesting results defined only for the binary case. Another work consists in transformating each non-binary constraint into a set of binary constraints, the obtained CSP is called primal. Unluckily, this transformation does not preserve satisfiability.

In this paper, we will propose some conditions, if they hold, a non-binary constraint can be decomposed into a set of binary constraints while preserving satisfiability. An experimental study proves that our approach is not artificial since some ternary benchmarks can be transformed into equivalent binary instances and effectively solved by MAC.

*Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet TUPLES (ANR-2010-BLAN-0210).

1 Introduction

Le *problème de satisfaction de contraintes* (CSP, [21]) constitue un formalisme important pour exprimer et résoudre efficacement plusieurs problèmes du monde réel. La majorité de ces problèmes s'expriment sous la forme d'instances CSP d'arité quelconque. Théoriquement, il est bien connu que toute instance d'arité quelconque peut être transformée en temps polynomial en instance binaire. Pour ce fait, nous avons principalement deux approches : soit en utilisant un des codages binaires connus tel que le codage dual [8], le codage par variable cachée [9] et le codage double [24] ou en convertissant (on dit aussi en décomposant) chaque contrainte non-binaire en un ensemble de contraintes binaires [8].

La première approche consiste à définir des nouvelles contraintes binaires sans convertir les contraintes originelles. Elle est basée sur les codages binaires, inspirés des représentations graphiques des instances non-binaires, pour obtenir une instance binaire équivalente sans décomposer aucune contrainte originelle (ou sa relation associée). Malheureusement, aucun de ces co-

dages n'a permis d'utiliser explicitement certaines propriétés intéressantes, comme la substitution et l'interchangeabilité [14] ou quelques autres qui portent sur l'identification de classes polynomiales ou l'application de certaines cohérences. En effet, appliquer certains niveaux de cohérences [25, 1] ou prouver l'appartenance à certaines classes polynomiales [6] de certaines instances non-binaires est souvent NP-difficile, ceci est peut être dû à la non-conversion de contraintes non-binaires. La seconde approche vise à décomposer toutes les contraintes non-binaires en un ensemble de contraintes binaires. Naturellement, cette approche n'a pas été développée par la suite vu qu'elle ne préserve pas la satisfiabilité.

Récemment, certains travaux ont montré l'intérêt de l'utilisation d'une version binaire des instances non-binaires d'un point de vue graphique [11]. En effet, la microstructure [19] d'une instance non-binaire exige l'utilisation des hypergraphes alors qu'une instance binaire peut être représentée par un graphe simple. Dans la littérature, la théorie des graphes s'avère être plus riche que celle des hypergraphes et l'utilisation de ces derniers semble être plus compliquée que les graphes. En plus, il est plus simple de calculer certains paramètres graphiques comme la largeur ou de vérifier quelques propriétés telle que l'acyclicité d'un graphe plutôt qu'un hypergraphe. Par ailleurs, certains autres travaux ont prouvé que l'extension de quelques classes polynomiales aux instances non-binaires peut être efficacement réalisée en utilisant les microstructures basées sur les codages binaires [10]. Cette tâche reste difficile à accomplir avec la définition de Cohen [3] qui s'appuie sur le complément d'un hypergraphe et qui ne se réfère pas aux codages binaires. En fait, cette notion pose plusieurs questions dont la principale consiste à savoir s'il faudrait considérer toutes les hyperarêtes qui correspondent aux relations universelles, à l'image de la notion de complémentaire de graphe dans le cas binaire. Mais dans ce cas, la taille de l'hypergraphe serait potentiellement exponentielle en fonction de la taille de l'instance.

Dans ce papier, nous nous intéressons à la conversion des contraintes non-binaires en contraintes binaires en préservant la satisfiabilité. Pour cela, nous allons dans un premier temps nous référer à la théorie des bases de données relationnelles pour définir une première règle pour la décomposition d'une contrainte en un ensemble de contraintes binaires sans perte de satisfiabilité. Pour le cas ternaire (instance d'arité trois), nous montrerons que chaque contrainte satisfaisant cette règle peut être remplacée par deux contraintes binaires. Dans un second temps, nous allons introduire une nouvelle règle, différente de la première, pour décomposer aussi les contraintes non-binaires, en préservant la cohérence. Pour le cas ternaire, une telle contrainte sera décomposée en trois contraintes binaires. Une partie de ce travail sera consacrée à étudier des propriétés autour de la décomposition, comme la complexité et la relation avec certaines classes polynomiales existantes.

Dans la section suivante, nous rappelons certaines définitions et notations nécessaires qui seront par la suite utilisées dans le reste de ce papier. Dans les sections 3 et 4, nous proposerons les deux règles permettant la décomposition des contraintes sans modifier la cohérence du problème de départ. En plus de certains résultats théoriques, notre étude est accompagnée de quelques résultats expérimentaux prouvant l'applicabilité de notre approche et certains liens avec les classes polynomiales. Avant de conclure, nous appliquons nos règles sur certaines contraintes bien particulières tel que les contraintes globales.

2 Préliminaires

Le *problème de satisfaction de contraintes* constitue un formalisme important pour exprimer et résoudre efficacement plusieurs problèmes réels en intelligence artificielle et en recherche opérationnelle.

Formellement, une *instance CSP* est définie comme suit :

Définition 1 (instance CSP) *Une instance CSP est un triplet $I = (V, D, C)$, où $V = \{V_1, ..., V_n\}$ est un ensemble fini de n* ***variables****, $D = \{D_1, ..., D_n\}$ est un ensemble fini de* ***domaines*** *contenant au plus d* ***valeurs****, un pour chaque variable et $C = \{C_1, ..., C_e\}$ est un ensemble de e* ***contraintes****. Chaque contrainte C_i est un couple $(S(C_i), R(C_i))$ avec :*

- *$S(C_i) = \{V_{i_1}, ..., V_{i_{n_i}}\} \subseteq V$, la* ***portée*** *de la contrainte,*
- *$R(C_i) \subseteq D_{i_1} \times ... \times D_{i_{a_i}}$, la* ***relation*** *qui autorise r tuples (compatibilité de valeurs).*

Nous supposons que toute variable apparaît au moins dans la portée d'une contrainte. $|S(C_i)|$ est l'**arité** de la contrainte C_i (c'est-à-dire, le nombre de variables sur lesquelles porte la contrainte c_i) et elle sera notée a_i . Si la contrainte est d'arité deux, elle est dite binaire et elle sera notée C_{ij} avec $S(C_{ij}) = \{V_i, V_j\}$. Si toutes les contraintes d'une instance I sont binaires alors I est dite **binaire**. Sinon (cas général), I est dite **non-binaire** (ou **d'arité quelconque**). Nous signalons que le cas ternaire est évoqué quand toutes les contraintes sont d'arité inférieure ou égale à trois.

Nous continuons avec les deux notations suivantes qui seront nécessaires pour la suite :

Notation 1 (projection d'un(e) tuple/relation) *Étant donnés une contrainte C_i, un tuple $t_i \in R(C_i)$*

et un ensemble de variables $\{V_{i_1}, ..., V_{i_k}\} \subseteq S(C_i)$ *:* $t_i[\{V_{i_1}, ..., V_{i_k}\}] = (v_j \in t_i \mid V_j \in \{V_{i_1}, ..., V_{i_k}\})$ *est la* ***projection*** *du tuple* t_i *sur* $\{V_{i_1}, ..., V_{i_k}\}$. $R(C_i)[\{V_{i_1}, ..., V_{i_k}\}] = \{t_i[\{V_{i_1}, ..., V_{i_k}\}] \mid t_i \in R(C_i)\}$ *est la projection de* $R(C_i)$ *sur* $\{V_{i_1}, ..., V_{i_k}\}$.

Notation 2 (restriction d'une contrainte)
Étant donnée une contrainte C_i, $C_i[\{V_{i_1}, ..., V_{i_k}\}]$ *est la restriction de* C_i *sur* $\{V_{i_1}, ..., V_{i_k}\}$ $(\subseteq S(C_i))$. *Si nous notons* $C_\ell = C_i[\{V_{i_1}, ..., V_{i_k}\}]$, *alors* $S(C_\ell) = \{V_{i_1}, ..., V_{i_k}\}$ *et* $R(C_\ell) = R(C_i)[\{V_{i_1}, ..., V_{i_k}\}]$.

Étant donnée une instance I, la question fondamentale est de décider si I a une solution (une affectation d'une valeur à chaque variable de I qui satisfait toutes les contraintes), problème bien connu comme étant NP-complet même pour le cas binaire.

Dans [8], les auteurs ont introduits une nouvelle méthode pour convertir une instance non-binaire en une instance binaire. Malheureusement, cette transformation ne préserve pas la satisfiabilité, c'est-à-dire l'ensemble de solutions de l'instance initiale n'est pas forcément égal à celui de l'instance transformée.

$R(C_\ell)$		
V_i	V_j	V_k
v_i	v_j	v_k
v'_i	v_j	v'_k
v'_i	v'_j	v_k

$R(C_{ij})$	$R(C_{jk})$	$R(C_{ik})$
$V_i\ V_j$	$V_j\ V_k$	$V_i\ V_k$
$v_i\ v_j$	$v_j\ v_k$	$v_i\ v_k$
$v'_i\ v_j$	$v_j\ v'_k$	$v'_i\ v'_k$
$v'_i\ v'_j$	$v'_j\ v_k$	$v'_i\ v_k$

FIGURE 1 – Une relation associée à une contrainte non-binaire $R(C_\ell)$ transformée en trois contraintes dont les relations sont $R(C_{ij})$, $R(C_{jk})$ et $R(C_{ik})$.

La figure 1 montre que les trois relations ($R(C_{ij})$,$R(C_{jk})$ et $R(C_{ik})$), obtenues après avoir décomposée la contrainte non-binaire C_ℓ, autorise l'affectation (v'_i, v_j, v_k) qui n'est pas autorisée par $R(C_\ell)$. Par conséquent, cette conversion ne préserve pas la satisfiabilité. Dans les sections suivantes, nous proposerons deux conditions pour décomposer les contraintes tout en préservant la satisfiabilité.

Pour des raisons de simplicité, nous supposons, dans la suite, que toutes les contraintes sont données en extension. Nous rappelons que toute contrainte en intension peut être transformée en une contrainte en extension.

3 Décomposition basée sur la dépendance multivaluée

Dans cette partie, nous illustrerons la première règle, basée sur la dépendance multivaluée, pour décomposer les contraintes non-binaires en contraintes binaires équivalentes. Dans la théorie des bases de données relationnelles, le concept de dépendance multivaluée a été initialement introduit par Fagin dans [13] pour décomposer et éliminer certaines redondances autorisées par la forme normale de Boyce Codd [2]. Du fait qu'une relation satisfait la dépendance multivaluée, elle est décomposable en deux relations sans perte de satisfiabilité. Ici, nous appliquons cette règle sur les instances non-binaires afin d'obtenir des instances binaires.

Nous commençons par une règle permettant de décomposer une contrainte C_ℓ en deux contraintes d'arité inférieure à a_ℓ. Avant cela, nous énonçons la notation suivante :

Notation 3 *Nous notons par* V_I *un sous-ensemble non-vide de* a_I *variables de* V $(\{V_{i_1}, ..., V_{i_{a_I}}\} \mid \forall 1 \leq k \leq a_I, V_{i_k} \in V\})$. v_I *note le tuple* $(v_{i_1}, ..., v_{i_{a_I}})$ *contenant une valeur pour chaque variable de* V_I.

Cette notation est nécessaire pour les contraintes d'arité quelconque.

Définition 2 (dépendance multivaluée [13])
Une contrainte d'arité quelconque C_ℓ *satisfait la dépendance multivaluée (MvD pour multivalued dependency) s'il existe trois sous-ensembles disjoints de variables* V_I, V_J *et* V_K *tel que* $S(C_\ell) = V_I \cup V_J \cup V_K$ *et* $V_I \twoheadrightarrow V_J, V_K$ *et* $\forall v_I \in R(C_\ell)[V_I], \forall v_J, v'_J \in R(C_\ell)[V_J]$ *et* $\forall v_K, v'_K \in R(C_\ell)[V_K]$ *tel que si*
- $(v_I, v_J, v_K) \in R(C_\ell)$
- $(v_I, v'_J, v'_K) \in R(C_\ell)$

alors,
- $(v_I, v'_J, v_K) \in R(C_\ell)$ *et*
- $(v_I, v_J, v'_K) \in R(C_\ell)$

Dans ce cas, nous pouvons noter $V_I \twoheadrightarrow V_J, V_K$. *Une instance non-binaire* $I = (V, D, C)$ *satisfait la dépendance multivaluée si chaque contrainte d'arité quelconque* $C_\ell \in C$ *satisfait MvD.*

Plus clairement, satisfaire MvD permet de décomposer une contrainte C_ℓ en deux contraintes équivalentes d'arité inférieure à a_ℓ.

Proposition 1 *Si une contrainte d'arité quelconque* C_ℓ *satisfait MvD, alors elle est décomposable en deux contraintes* C'_ℓ *et* C''_ℓ *d'arité inférieure à* a_ℓ *en préservant la satisfiabilité.*

Preuve : (par l'absurde) Soit C_ℓ une contrainte d'arité quelconque qui satisfait MvD, nous prouvons par l'absurde que la décomposition de contraintes MvD préserve la satisfiabilité. Donc, nous supposons qu'il existe une affectation $\mathcal{A}$ qui ne viole aucune des nouvelles contraintes (obtenues après décomposition) sans satisfaire la contrainte originelle C_ℓ. Comme C_ℓ satisfait

MvD, il existe trois sous-ensembles disjoints de variables V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$ et $V_I \twoheadrightarrow V_J, V_K$. Ainsi, nous pouvons exprimer $\mathcal{A}$ comme (v_I, v_J, v_K). Comme $\mathcal{A}$ viole C_ℓ, nous devons évidemment avoir deux tuples t_1 et t_2 appartenant à $R(C_\ell)$ tel que

- $t_1[V_I] = v_I$, $t_1[V_J] = v_J$ et $t_1[V_K] = v'_K$,
- $t_2[V_I] = v_I$, $t_2[V_J] = v'_J$ et $t_2[V_K] = v_K$.

Nous précisons que v_K doit absolument être différent de v'_K (pareillement pour v_J et v'_J) sinon $\mathcal{A}$ ne viole pas C_ℓ. Dans cette direction, nous avons

- $(v_I, v_J, v'_K) \in R(C_\ell)$
- $(v_I, v'_J, v_K) \in R(C_\ell)$

Or, par définition de MvD, nous aurons

- $(v_I, v_J, v_K) \in R(C_\ell)$ (1) et
- $(v_I, v'_J, v'_K) \in R(C_\ell)$ (2).

Finalement, (1) contredit notre hypothèse. Par conséquent, la décomposition de contraintes MvD préserve la satisfiabilité. □

Sémantiquement, une contrainte C_ℓ est décomposable s'il existe deux sous-ensembles de variables non-disjoints $X, Y \subsetneq S(C_\ell)$ tel que $R(C_\ell)$ est la jointure de $R(C_\ell)[X]$ et $R(C_\ell)[Y]$.

Exemple 1 *Cet exemple illustre le cas d'une contrainte C_ℓ d'arité quatre et sa relation associée. En considérant $V_I = \{V_i, V_m\}$, $V_J = \{V_j\}$ et $V_K = \{V_k\}$, C_ℓ satisfait MvD, donc elle est décomposable en deux contraintes C'_ℓ et C''_ℓ. Les tables ci-dessous représentent les relations $R(C_\ell)$, $R(C'_\ell)$ et $R(C''_\ell)$.*

$R(C_\ell)$			
V_i	V_m	V_j	V_k
v_i	v_m	v_j	v'_k
v_i	v_m	v'_j	v_k
v_i	v_m	v'_j	v'_k
v_i	v_m	v_j	v_k
v'_i	v'_m	v_j	v_k

$R(C'_\ell)$	$R(C''_\ell)$
$V_i\ V_m\ V_j$	$V_i\ V_m\ V_k$
$v_i\ v_m\ v_j$	$v_i\ v_m\ v_k$
$v_i\ v_m\ v'_j$	$v_i\ v_m\ v'_k$
$v'_i\ v'_m\ v_j$	$v'_i\ v'_m\ v_k$

Pour le cas ternaire, satisfaire la dépendance multivaluée garantie la décomposition de la contrainte en deux contraintes binaires sans perte de satisfiabilité.

D'une façon générale, être MvD permet la décomposition d'une contrainte d'arité quelconque en deux contraintes d'arité inférieure à celle de la contrainte originelle et qui ne sont pas forcément toutes les deux binaires. Pour cela, nous définissons une nouvelle forme récursive de la dépendance multivaluée qui permettra de décomposer les contraintes d'arité quelconque en un ensemble de contraintes binaires équivalentes.

Définition 3 (dépendance multivaluée décrémentale) *Une contrainte C_ℓ d'arité $a_\ell \geq 3$ satisfait la dépendance multivaluée décrémentale (DMvD pour decremental multivalued dependency) si*

1. *il existe trois sous-ensembles disjoints de variables V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$ et C_ℓ satisfait MvD*
2. *$a_I + a_J \geq 3$ alors $C_\ell[V_I \cup V_J]$ satisfait DMvD.*
3. *$a_I + a_K \geq 3$ alors $C_\ell[V_I \cup V_K]$ satisfait DMvD.*

Une instance non-binaire $I = (V, D, C)$ satisfait la dépendance multivaluée décrémentale si chaque contrainte non-binaire $C_\ell \in C$ est DMvD.

Nous devons maintenant montrer que satisfaire DMvD permet à la contrainte d'être remplacée par un ensemble de contraintes binaires équivalentes. Intuitivement, et d'après la définition précédente, si une contrainte C_ℓ satisfait DMvD, elle sera récursivement décomposée selon la proposition 1 en commençant par décomposer C_ℓ en deux contraintes équivalentes d'arité inférieure à celle de C_ℓ et nous continuons avec les nouvelles contraintes jusqu'à l'obtention de contraintes binaires. Le théorème suivant montre que cette règle de décomposition préserve la satisfiabilité.

Théorème 1 *Si une contrainte C_ℓ est DMvD, alors elle est décomposable en un ensemble de contraintes binaires sans perte de satisfiabilité.*

Preuve : (par induction) Il est clair que DMvD est définie récursivement en respectant MvD. Donc, nous allons prouver par induction que quelles que soient la contrainte C_ℓ et son arité a_ℓ, si C_ℓ satisfait DMvD alors elle sera décomposée en un ensemble de contraintes binaires sans perte de satisfiabilité.

- **cas de base :** pour $a_\ell = 3$, nous pouvons constater d'après le théorème 1 que si une contrainte ternaire satisfait MvD, alors elle est décomposable en deux contraintes d'arité inférieure (obligatoirement 2) sans perte de satisfiabilité.
- **hypothèse d'induction :** nous supposons que les contraintes DMvD avec une arité $a_\ell \leq p - 1$ sont décomposables en un ensemble de contraintes binaires sans perte de satisfiabilité.
- **cas inductif :** nous prouvons que c'est le cas aussi quand $a_\ell = p$. Comme C_ℓ est DMvD, elle est donc décomposable en deux contraintes C'_ℓ et C''_ℓ sans perte de satisfiabilité, chacune a une arité inférieure à p. Nous avons aussi supposé que lorsqu'une contrainte satisfait DMvD et son arité est inférieure à p, elle est décomposable en un ensemble de contraintes binaires en préservant la satisfiabilité.

DMvD préserve la satisfiabilité quelle que soit l'arité de la contrainte. □

Nous étudions maintenant la complexité de vérification de cette propriété et nous montrons qu'elle est

polynomiale seulement quand les trois sous-ensembles disjoints V_I, V_J et V_K sont donnés à l'avance.

Proposition 2 *Étant donnés une contrainte C_ℓ et trois sous-ensembles disjoints de variables V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$, vérifier si C_ℓ satisfait DMvD par rapport aux trois sous-ensembles disjoints V_I, V_J et V_K peut être réalisé en temps polynomial.*

Preuve : Pour trois sous-ensembles disjoints de variables V_I, V_J et V_K donnés tel que $S(C_\ell) = V_I \cup V_J \cup V_K$, nous aurons besoin de $O(a_\ell.r^2.log(r))$ pour vérifier si la contrainte C_ℓ satisfait DMvD.

- r^2 pour énumérer les tuples,
- $log(r)$ pour tester l'appartenance des tuples à la relation associée à la contrainte,
- a_ℓ ou plus précisément $a_\ell - 2$ pour appliquer récursivement le test sur les nouvelles contraintes.

La complexité de la décomposition est $O(r.a_\ell^2)$:

- r pour parcourir les tuples de la contrainte C_ℓ,
- a_ℓ^2 ou plus précisément $a_\ell(a_\ell-1)/2$ qui correspond au nombre maximal de contraintes binaires.

Par conséquent, vérifier si une contrainte satisfait DMvD et par la suite la décomposer est réalisable en temps polynomial. □

La contrainte C_ℓ de l'exemple 1 satisfait DMvD parce que C'_ℓ et C''_ℓ sont d'arité trois et sont MvD.

Pour le cas ternaire, cette propriété peut évidemment être vérifiée en temps polynomial et il existe un algorithme (Algorithme 1) en $O(er^2.log(r))$ pour vérifier si une instance ternaire I pourrait être transformée en une instance binaire équivalente en respectant la propriété MvD. La décomposition d'une contrainte MVD nécessite un temps linéaire $O(r)$ ($O(er)$ pour décomposer toutes les contraintes).

Algorithm 1: Tester si les contraintes d'une instance ternaire sont décomposables

```
function TESTER_DECOMPOSITION(I = (V, D, C) : CSP) : Booléen
    foreach C_ℓ ∈ C avec S(C_ℓ) = {V_i, V_j, V_k} do
        if (not TESTER_CONTRAINTE(C_ℓ,V_i, V_j, V_k)) et
      (not TESTER_CONTRAINTE(C_ℓ,V_j, V_i, V_k)) et (not
      TESTER_CONTRAINTE(C_ℓ,V_k, V_j, V_i)) then
            return faux
    return vrai
end function
function TESTER_CONTRAINTE( C_ℓ : Contrainte, V_i, V_j, V_k :
Variable) : Booléen
for t_ℓ, t'_ℓ ∈ R(C_ℓ) avec t_ℓ[{V_i}] = t'_ℓ[{V_i}] do
        if ((t_ℓ[{V_j}] ≠ t'_ℓ[{V_j}]) et (t_ℓ[{V_k}] ≠ t'_ℓ[{V_k}]))
    then
            if (((t_ℓ[{V_i}], t'_ℓ[{V_j}], t_ℓ[{V_k}]) ∉ R(C_ℓ)) ou
        ((t_ℓ[{V_i}], t_ℓ[{V_j}], t'_ℓ[{V_k}]) ∉ R(C_ℓ))) then
                return faux
    return vrai
end function
```

Nous pouvons immédiatement constater qu'une nouvelle classe polynomiale pourrait être définie pour les instances ternaires bivalentes [1].

Proposition 3 *La classe des instances ternaires bivalentes qui satisfont MvD est polynomiale.*

Preuve : Dans [15, 4], une classe polynomiale pour CSP est un ensemble (fini ou infini) d'instances pour lesquelles il existe deux algorithmes de complexité polynomiale, un premier pour la reconnaissance d'instances et un second pour les résoudre. Comme nous l'avons mentionné auparavant, $O(er^2.log(r))$ est suffisant pour vérifier si une instance ternaire satisfait la MvD. Pour la résolution, nous commençons par convertir l'instance en instance binaire, ce qui nécessite $O(er)$ en temps. Ensuite, nous savons que les instances binaires bivalentes définissent une classe polynomiale [7] dont la cohérence de chemin est une procédure de décision. Donc, il faut $O(n^3d^3)$ pour appliquer PC-4 [17] afin d'avoir une instance globalement cohérente [7]). □

Nous avons effectué une étude expérimentale sur 701 instances ternaires de la la compétition CP [2] pour tester l'applicabilité de notre approche en pratique. Finalement, nous avons obtenu un total de résultats pour 581 instances, nous précisons que nous avons fixé une durée d'une heure pour chaque benchmark et que notre bibliothèque ne couvre pas les instances avec des contraintes globales. 72 benchmarks ont été détectés comme étant MvD parmi lesquels nous pouvons citer les familles `pseudo/primeDimacs`, `pseudo/par`, `pseudo/garden` et `pseudo/aim`. Nous notons que tous ces benchmarks appartiennent à la classe polynomiale définie dans la proposition 3. De plus, la version transformées de ces benchmarks est mieux résolue par MAC[22] que celui de la version ternaire (y compris le temps de décomposition).

La table 1 montre quelques autres résultats des benchmarks donts les contraintes ne respectent pas toutes MvD. Les notations suivantes signifient :

- Family : nom de famille de benchmarks
- N : nombre d'instances ternaires testées de cette famille
- n : nombre de variables
- e/e' : nombre de contraintes / nombre de contraintes ternaires
- E : nombre de contraintes décomposables.

Pour le cas général, si nous ne connaissons pas la répartition de variables en trois sous-ensembles disjoints, la vérification de cette propriété semble être plus complexe.

1. Une instance CSP est dite bivalente si la taille de tous les domaines de variables est inférieure ou égale à deux.

2. voir http ://www.cril.univ-artois.fr/CPAI08 pour plus de détails.

Family	N	n	e/e'	E
ssa	1	757	847/479	460
aim-100	24	100	263/261	136
primes-*	16	100	46/23	16
travellingSalesman-*	30	69	290/23	1

TABLE 1 – Résultats expérimentaux de certains benchmarks ternaires montrant le nombre de contraintes MvD.

Théorème 2 *Étant donnée une contrainte C_ℓ, vérifier si C_ℓ satisfait DMvD est NP-complet.*

Preuve : Non détaillée par manque d'espace. Nous spécifions que la preuve s'appuie sur une réduction polynomiale du problème 3-SAT connu pour être NP-complet. □

Nous pouvons aussi définir un niveau plus élevé de MvD pour les contraintes non-binaires.

Définition 4 (dépendance multivaluée forte) *Étant donnée une contrainte C_ℓ avec $a_\ell \geq 3$, nous disons que C_ℓ satisfait la dépendance multivaluée forte (SMvD pour strong multivalued dependency) si quels que soient les trois sous-ensembles disjoints de variables V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$, nous avons C_ℓ satisfait DMvD. Une instance non-binaire $I = (V, D, C)$ satisfait la dépendance multivaluée forte si chaque contrainte non-binaire $C_\ell \in C$ est SMvD.*

Revenons à l'exemple 1, la contrainte C_ℓ n'est pas SMvD car si on considère $V_I = \{V_j, V_k\}$, $V_J = \{V_i\}$ et $V_K = \{V_m\}$, C_ℓ ne satisfait pas DMvD.

Avant de conclure cette section, nous devons préciser les points suivants. Pour une contrainte C_ℓ avec $S(C_\ell) = V_I \cup V_J \cup V_K$, si nous n'avons pas $V_I \twoheadrightarrow V_J, V_K$ nous ne pouvons rien infirmer à propos de $V_J \twoheadrightarrow V_I, V_K$ et $V_K \twoheadrightarrow V_I, V_J$. Nous signalons également que si $V_J \twoheadrightarrow V_I, V_K$ et $V_K \twoheadrightarrow V_I, V_J$ nous ne pouvons rien confirmer pour $V_I \twoheadrightarrow V_J, V_K$.

4 Décomposition basée sur l'interdépendance

Dans cette section, nous présentons une nouvelle règle, que nous appelons *interdépendance*, permettant la décomposition de contraintes non-binaires sans perte de satisfiabilité. L'interdépendance est définie comme suit :

Définition 5 (interdépendance) *Une contrainte C_ℓ d'arité $a_\ell \geq 3$ satisfait l'interdépendance (ID pour interdependency) s'il existe trois sous-ensembles disjoints de variables V_I, V_J et V_K tel que $\forall v_I \in R(C_\ell)[V_I], \forall v_J, v'_J \in R(C_\ell)[V_J]$ et $\forall v_K, v'_K \in R(C_\ell)[V_K]$ si*
- $(v_I, v_J, v_K) \in R(C_\ell)$
- $(v_I, v'_J, v'_K) \in R(C_\ell)$

alors il n'existe aucun v'_I tel que
- $(v'_I, v'_J, v_K) \in R(C_\ell)$ *ou*
- $(v'_I, v_J, v'_K) \in R(C_\ell)$

Dans ce cas, nous disons aussi que V_I, V_J et V_K sont interdépendants (ID). Une instance $I = (V, D, C)$ satisfait l'interdépendance si chaque contrainte non-binaire $C_\ell \in C$ satisfait ID.

Comme pour la dépendance multivaluée, une contrainte C_ℓ dont la portée peut être partitionnée en trois sous-ensembles disjoints de variables interdépendants est décomposable en trois contraintes d'arité inférieure à celle de la contrainte originelle.

Théorème 3 *Si une contrainte d'arité quelconque C_ℓ satisfait ID, alors elle est décomposable en trois contraintes $C_\ell[V_I]$, $C_\ell[V_J]$ et $C_\ell[V_K]$ sans perte de satisfiabilité.*

Preuve : (par l'absurde) Soit C_ℓ une contrainte d'arité quelconque qui satisfait ID, nous prouvons par l'absurde que la décomposition de contraintes ID préserve la satisfiabilité. Donc, nous supposons qu'il existe une affectation $\mathcal{A}$ qui ne viole aucune nouvelles contraintes (obtenues après décomposition) sans satisfaire la contrainte originelle C_ℓ. Donc, il existe trois sous-ensembles disjoints de variables V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$ et C_ℓ satisfait ID. Dans ce cas, nous pouvons exprimer $\mathcal{A}$ comme (v_I, v_J, v_K). Comme $\mathcal{A}$ viole C_ℓ, il existe trois tuples t_1, t_2 et t_3 appartenant à $R(C_\ell)$ tel que
- $t_1[V_I] = v_I$, $t_1[V_J] = v_J$ et $t_1[V_K] = v'_K$ (1),
- $t_2[V_I] = v_I$, $t_2[V_J] = v'_J$ et $t_2[V_K] = v_K$ (2),
- $t_3[V_I] = v'_I$, $t_3[V_J] = v_J$ et $t_3[V_K] = v_K$ (3).

Nous précisons que v_I doit être différent de v'_I (pareillement pour v_J, v_K et v'_J, v'_K). Autrement, $\mathcal{A}$ ne viole pas C_ℓ. Dans ce cas, nous pouvons écrire
- $(v_I, v_J, v'_K) \in R(C_\ell)$
- $(v_I, v'_J, v_K) \in R(C_\ell)$

et par définition de ID, il n'existe aucun $v'_I \neq v_I$ tel que
- $(v'_I, v_J, v_K) \in R(C_\ell)$ (a)

ce qui se contredit avec (3) et donc la décomposition de contraintes ID préserve la satisfiabilité. □

Exemple 2 *Ce deuxième exemple illustre le cas d'une contrainte C_ℓ d'arité quatre et sa relation associée. En considérant $V_I = \{V_i\}$, $V_J = \{V_m, V_j\}$ et $V_K = \{V_k\}$, C_ℓ satisfait ID, donc elle est décomposable en trois contraintes C'_ℓ, C''_ℓ et C'''_ℓ. Les tables ci-dessous représentent les quatre relations $R(C_\ell)$, $R(C'_\ell)$, $R(C''_\ell)$ et $R(C'''_\ell)$.*

$R(C_\ell)$			
V_i	V_m	V_j	V_k
v_i	v'_m	v'_j	v_k
v_i	v_m	v_j	v'_k
v'_i	v_m	v'_j	v'_k

$R(C'_\ell)$	$R(C''_\ell)$	$R(C'''_\ell)$
$V_i\ V_m\ V_j$	$V_m\ V_j\ V_k$	$V_i\ V_k$
$v_i\ v'_m\ v'_j$	$v'_m\ v'_j\ v_k$	$v_i\ v_k$
$v_i\ v_m\ v_j$	$v_m\ v_j\ v'_k$	$v_i\ v'_k$
$v'_i\ v_m\ v'_j$	$v_m\ v'_j\ v'_k$	$v'_i\ v'_k$

Malheureusement, l'interdépendance ne garantie pas la décomposition de contraintes non-binaires en contraintes binaires équivalentes. Pour cela, nous introduisons l'interdépendance décrémentale.

Définition 6 (interdépendance décrémentale) *Étant donnée une contrainte C_ℓ avec $a_\ell \geq 3$, nous disons que C_ℓ satisfait l'interdépendance décrémentale (DID pour decremental interdependency) si*

1. *il existe une partition V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$ et C_ℓ satisfait ID*
2. *$a_I + a_J \geq 3$ alors $C_\ell[V_I \cup V_J]$ satisfait DID.*
3. *$a_I + a_K \geq 3$ alors $C_\ell[V_I \cup V_K]$ satisfait DID.*
4. *$a_J + a_K \geq 3$ alors $C_\ell[V_J \cup V_K]$ satisfait DID.*

Une instance non-binaire $I = (V, D, C)$ satisfait l'interdépendance décrémentale si chaque contrainte non-binaire $C_\ell \in C$ est DID.

Nous pouvons maintenant déduire que DID est une condition suffisante pour décomposer une contrainte non-binaire en un ensemble de contraintes binaires équivalentes. Comme nous l'avons mentionné dans la définition précédente, la contrainte C_ℓ sera récursivement décomposée en respectant la proposition 3.

Théorème 4 *Si une contrainte C_ℓ satisfait DID, alors elle sera décomposée en un ensemble de contraintes binaires sans perte de satisfiabilité.*

Preuve : (par induction) Il est clair que DID est récursivement définie et basée sur ID. Nous prouvons par induction que quelle que soit l'arité a_ℓ de la contrainte C_ℓ la décomposition de contraintes qui satisfont DID préserve la satisfiabilité.

- **cas de base :** pour $a_\ell = 3$, nous avons montré dans le théorème 3 que la décomposition de contraintes ID préserve la satisfiabilité.
- **hypothèse d'induction :** nous supposons que la satisfiabilité est préservée pour $a_\ell \leq p - 1$.
- **cas inductif :** nous prouvons qu'elle restera préservée quand $a_\ell = p$. Comme C_ℓ satisfait DID, elle est donc décomposable en trois contraintes binaires $C_\ell[V_I \cup V_J]$, $C_\ell[V_J \cup V_K]$ et $C_\ell[V_I \cup V_K]$ sans perte de satisfiabilité, chacune de ses nouvelles contraintes a une arité inférieure à p. Nous avons supposé que lorsqu'une contrainte satisfait DID et son arité est inférieure à p, elle est récursivement décomposable en un ensemble de contraintes binaires sans perte de satisfiabilité.

La décomposition de contraintes DID préserve la satisfiabilité quelle que soit l'arité de la contrainte. □

Nous pouvons constater que la contrainte C_ℓ de l'exemple 2 est DID car les deux contraintes d'arité trois C'_ℓ et C''_ℓ satisfont évidemment ID.

Étant donnés une contrainte C_ℓ et trois sous-ensembles disjoints de variables V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$, vérifier si C_ℓ satisfait DID par rapport à V_I, V_J et V_K peut se réaliser en temps polynomial. Par contre, dans le cas général, déterminer s'il existe trois sous-ensembles disjoints V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$ est NP-complet.

Proposition 4 *Étant donnés une contrainte C_ℓ et trois sous-ensembles disjoints de variables V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$, vérifier si C_ℓ satisfait DID par rapport à V_I, V_J et V_K peut se réaliser en temps polynomial.*

Preuve : Similaire à la preuve de la proposition 2 □.

Théorème 5 *Étant donnée une contrainte C_ℓ, tester s'il existe trois sous-ensembles disjoints de variables V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$ et V_I, V_J et V_K sont DID est NP-complet.*

Preuve : Supprimée pour manque d'espace. comme pour le théorème 2, nous devons encore spécifier que cette preuve repose sur une réduction polynomiale du problème 3-SAT bien connu pour être NP-complet. □

Contrairement à la dépendance multivaluée, l'interdépendance est symétrique, c'est-à-dire si une contrainte C_ℓ avec $S(C_\ell) = \{V_i, V_j, V_k\}$ est ID par rapport à la variable V_i, elle le sera aussi par rapport aux variables V_j et V_k. Par conséquent, nous n'avons pas à tester la propriété pour les trois variables, il suffit d'avoir un résultat d'une variable de $S(C_\ell)$.

Pour le cas ternaire, cette propriété peut être vérifiée, pour une contrainte donnée, en temps polynomial ($O(d.r^2 log(r))$) alors que la décomposition d'une telle contrainte peut se réaliser en temps linéaire ($O(r)$). Ceci nous conduit à la proposition suivante.

Proposition 5 *La classe d'instances ternaires bivalentes qui satisfont l'interdépendance est polynomiale.*

Preuve : Similaire à la preuve de la proposition 3. □

D'un point de vue expérimental, tous les benchmarks qui satisfont MvD sont aussi ID en plus de certaines autres instances de la famille `primes-20` et `primes-30`. Toutes ces instances satisfont soit la classe polynomiale décrite dans la proposition 5, soit BTP [3]

3. Une instance binaire I satisfait **Broken Triangle Property (BTP)** par rapport à un ordre sur les variables $<$ si, pour tout triplet de variables (V_i, V_j, V_k) tel que $V_i < V_j < V_k$, si $(v_i, v_j) \in R(C_{ij})$, $(v_i, v'_k) \in R(C_{ik})$ et $(v_j, v''_k) \in R(C_{jk})$, alors soit $(v_i, v''_k) \in R(C_{ik})$, soit $(v_j, v'_k) \in R(C_{jk})$.

Family	N	n	e/e'	E
ssa	1	757	847/479	273
pret	8	105	70/70	35
dubois	13	98	65/65	64
travellingSalesman-*	30	69	290/23	1

TABLE 2 – Résultats expérimentaux sur quelques benchmarks ternaires montrant le nombre de contraintes ID.

[5] ou soit DBTP [4] [10]. De plus, les instances binaires obtenues sont mieux résolues par MAC que les instances ternaires originelles.

Pour quelques instances appartenant à `dubois`, `pret` et `primes`, toutes les contraintes ternaires sauf une ont été remplacées par des contraintes binaires. Globalement, nous avons réussi à convertir 86 benchmarks de 581 considérés (ce qui représente environ 14% au total).

Maintenant, nous définissons l'interdépendance forte.

Définition 7 (interdépendance forte) *Étant donnée une contrainte C_ℓ avec $a_\ell \geq 3$, nous disons que C_ℓ satisfait l'interdépendance forte (SID pour strong interdependency) si pour chaque trois sous-ensembles disjoints de variables V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$, nous avons C_ℓ satisfait DID. Une instance non-binaire $I = (V, D, C)$ satisfait l'interdépendance forte si chaque contrainte non-binaire $C_\ell \in C$ satisfait SID.*

Pour l'exemple 2, la contrainte C_ℓ ne satisfait pas l'interdépendance forte en considérant $V_I = \{V_m, V_k\}$, $V_J = \{V_j\}$ et $V_K = \{V_i\}$.

Pour finir, nous définissons comment une contrainte pourrait être parfaitement décomposable.

Définition 8 (décomposition parfaite) *Une contrainte d'arité quelconque C_ℓ est* parfaitement décomposable *si elle satisfait l'interdépendance pour chaque trois sous-ensembles disjoints V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$.*

Nous montrons que les contraintes parfaitement décomposables possèdent une propriété intéressante qui sera utilisée dans la suite.

Lemme 1 *Une contrainte d'arité quelconque C_ℓ satisfait ID pour chaque triplet de sous-ensembles disjoints de variables V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$, si et seulement si, $C_\ell[\{V_i, V_j, V_k\}]$ satisfait ID pour chaque triplet de variables V_i, V_j et $V_k \in S(C_\ell)$.*

4. DBTP est une extension de BTP aux instances d'arité quelconque en utilisant le codage dual.

Preuve : ($\Rightarrow$) Pour une contrainte C_ℓ, nous supposons qu'elle satisfait ID tous trois sous-ensembles disjoints de variables V_I, V_J et V_K tel que $S(C_\ell) = V_I \cup V_J \cup V_K$ alors qu'il existe V_i, V_j et $V_k \in S(C_\ell)$ tel que $C_\ell[\{V_i, V_j, V_k\}]$ ne satisfait pas ID. Donc, il existe $v_i, v'_i \in D_i$, $v_j, v'_j \in D_j$ et $v_k, v'_k \in D_k$ tel que
- $(v_i, v_j, v_k) \in R(C_\ell)[\{V_i, V_j, V_k\}]$
- $(v_i, v'_j, v'_k) \in R(C_\ell)[\{V_i, V_j, V_k\}]$ et
- $(v'_i, v'_j, v_k) \in R(C_\ell)[\{V_i, V_j, V_k\}]$ (ou $(v'_i, v_j, v'_k) \in R(C_\ell)[\{V_i, V_j, V_k\}]$).

Dans ce cas, il est obligatoire que v_i du premier tuple (v_i, v_j, v_k) appartienne à v_I et v_i du second tuple (v_i, v'_j, v'_k) appartienne à v'_I qui est différent de v_I, autrement V_I, V_J et V_K ne sont pas ID. Pour cela, il existe $v_m, v'_m \in D_m$ ($V_m \in V_I$) tel que $v_I[\{V_i, V_m\}] = (v_i, v_m)$ et $v'_I[\{V_i, V_m\}] = (v'_i, v_m)$. Si nous considérons une autre répartition, c'est-à-dire $V_I = V_I - \{V_m\}$ et $V_J = V_J \cup \{V_m\}$, V_I, V_J et V_K ne sont pas ID ce qui est impossible car cela contredit notre hypothèse.
($\Leftarrow$) Supposons pour une contrainte donnée C_ℓ qu'il existe trois sous-ensembles disjoints de variables V_I, V_J et V_K (avec $S(C_\ell) = V_I \cup V_J \cup V_K$) qui ne sont pas ID bien que V_i, V_j et V_k sont ID pour tout $V_i, V_j, V_k \in S(C_\ell)$. Alors, $\exists v_I, v'_I \in R(C_\ell)[V_I]$, $v_J, v'_J \in R(C_\ell)[V_J]$, v_K, $v'_K \in R(C_\ell)[V_K]$ de telle façon que
- $(v_I, v_J, v_K) \in R(C_\ell)$,
- $(v_I, v'_J, v'_K) \in R(C_\ell)$ et
- $(v'_I, v'_J, v_K) \in R(C_\ell)$ (ou $(v'_I, v_J, v'_K) \in R(C_\ell)$)

$v_I \neq v'_I \Longrightarrow \exists V_i \in V_I$ tel que $v_I[\{V_i\}] \neq v'_I[\{V_i\}]$, nous notons $v_i = v_I[\{V_i\}]$ et $v'_i = v'_I[\{V_i\}]$. D'une façon similaire, nous obtenons $v_j = v_J[\{V_j\}]$, $v'_j = v'_J[\{V_j\}]$, $v_k = v_K[\{V_k\}]$ et $v'_k = v'_K[\{V_k\}]$. Ceci implique que $(v_i, v_j, v_k) \in R(C_\ell)[\{V_i, V_j, V_k\}]$, $(v_i, v'_j, v'_k) \in R(C_\ell)[\{V_i, V_j, V_k\}]$ et $(v'_i, v'_j, v_k) \in R(C_\ell)[\{V_i, V_j, V_k\}]$. Par conséquent, V_i, V_j et V_k ne sont pas ID ce qui contredit notre hypothèse. □

Nous établissons maintenant le lien entre l'interdépendance forte et la décomposition parfaite.

Corollaire 1 *Si une contrainte C_ℓ satisfait l'interdépendance forte, alors elle est parfaitement décomposable.*

5 Comparaison entre les deux règles

Ici, nous prouvons que les deux règles de décomposition sont différentes et qu'il n'existe aucun lien entre les deux approches.

Proposition 6 *L'interdépendance et la dépendance multivaluée sont incomparables.*

Preuve : Nous devons juste noter que certaines contraintes de la famille de benchmarks `dubois` sont décomposables en considérant l'interdépendance mais

pas en considérant la dépendance multivaluée. Certains autres (comme `ssa`) ont des contraintes qui sont décomposables en utilisant la dépendance multivaluée mais pas en utilisant l'interdépendance. Finalement, nous avons certains benchmarks qui sont décomposables en considérant les deux règles (comme `primes-Dimacs`). □

Nous pouvons finalement signaler que même lorsqu'une contrainte C_ℓ avec $S(C_\ell) = V_I \cup V_J \cup V_K$ est décomposable en considérant MvD et par rapport à V_I, V_J et V_K, ceci n'impliquera rien sur la décomposition en considérant ID.

6 Travaux connexes

D'une part, nous pouvons constater que la notion de décomposition traitée dans ce papier pourrait être comparée à la notion de transformation introduite dans [12] sauf que cette dernière, telle qu'elle est définie, n'autorise pas la décomposition de contraintes. Certains autres travaux [16, 18] ont étudiés auparavant la décomposabilité de contraintes mais en se basant sur des règles différentes inspirées dans certains cas de la théorie des bases de données relationnelles. Nous devons aussi signaler que les propositions 3 et 5 sont forcément des cas particuliers des classes de Schaefer [23], modulo renommage des valeurs.

D'autre part, plusieurs autres travaux ont étudié la décomposition et en particulier pour les contraintes globales. Pour des raisons d'espace, nous nous limitons ici aux contraintes *All-Diff* [20]. Pour cela, nous supposons que toutes les variables possèdent le même domaine (et par conséquent $v_i = v_j = v_k$ et $v'_i = v'_j = v'_k$, etc.). Nous montrons que les contraintes All-Diff sont décomposables en considérant l'interdépendance et pas la dépendance multivaluée.

Théorème 6 *Toute contrainte non-binaire All-Diff est parfaitement décomposable.*

Preuve : Pour le cas ternaire, toute contrainte C_ℓ avec $S(C_\ell) = \{V_i, V_j, V_k\}$ est parfaitement décomposable car si nous avons $(v_i, v'_j, v''_k) \in R(C_\ell)$ et $(v_i, v''_j, v'_k) \in R(C_\ell)$ alors il est impossible que $v'''_i \in D_i$ existe tel que $(v'''_i, v'_j, v'_k) \in R(C_\ell)$ et $(v'''_i, v''_j, v''_k) \in R(C_\ell)$ vu que v'_j et v'_k sont égaux et ils ne peuvent pas apparaître dans un même tuple (par définition de contrainte All-Diff). Pour le cas général, supposons qu'il est possible d'avoir une contrainte All-Diff non-binaire qui n'est pas parfaitement décomposable. D'après le lemme 1, il existe V_i, V_j, et $V_k \in S(C_\ell)$ tel que V_i, V_j et V_k ne sont pas ID. Puisque $R(C_\ell)[\{V_i, V_j, V_k\}]$ est aussi All-Diff, il en résulte que cette contrainte ternaire All-Diff est parfaitement décomposable, ce qui est impossible. □

Théorème 7 *Les contraintes All-Diff ne sont pas décomposable en considérant la dépendance multivaluée.*

Preuve : Étant donnée une contrainte C_ℓ avec $S(C_\ell) = \{V_i, V_j, V_k\}$, si nous avons $(v_i, v'_j, v''_k) \in R(C_\ell)$ et $(v_i, v''_j, v'_k) \in R(C_\ell)$, nous devons avoir $(v_i, v'_j, v'_k) \in R(C_\ell)$ et $(v_i, v''_j, v''_k) \in R(C_\ell)$ pour que C_ℓ soit décomposable. Mais nous savons que ceci est impossible à cause de l'affectation d'une même valeur à des variables différentes dans un tuple (par définition des contraintes All-Diff). □

Enfin, nous signalons que certaines contraintes globales paramétrées comme AtMost et AtLeast ne satisfont ni la dépendance multivaluée ni l'interdépendance.

7 Conclusion

Dans ce papier, nous avons introduit, dans un premier temps, deux nouvelles règles pour tester si une contrainte non-binaire est décomposable en un ensemble de contraintes binaires (ou non-binaires d'arité inférieure à celle de la contrainte originelle) sans perte de satisfiabilité. La première règle est basée sur une notion qui a été précédemment introduite dans la théorie des bases de données relationnelles. La deuxième s'est appuyée sur le concept d'interdépendance entre les valeurs de variables appartenant à la porté de la contrainte. Notre étude est accompagnée d'une expérimentation qui montre l'applicabilité de ces deux règles sur les instances ternaires surtout qu'elles appartiennent toutes à des classes polynomiales connues. Dans un second temps, nous avons montré que la dépendance multivaluée et l'interdépendance sont différentes. Ensuite, nous les avons testées sur quelques contraintes globales avant de les comparer à certains travaux précédents. Dans le futur, plusieurs pistes méritent d'être étudiées. Tout d'abord, il faut tester si notre approche peut être appliquée sur certaines contraintes globales de l'état de l'art autre que les contraintes All-Diff. Ensuite, une deuxième piste consiste à savoir si les instances transformées en considérant une de nos deux approches sont incluses dans une classe polynomiale connue ou si elles définissent une nouvelle classe traitable.

8 Remerciements

Je tiens à remercier Philippe Jégou, Cyril Terrioux et les relecteurs anonymes de cet article qui ont contribué à l'amélioration de la rédaction et qui ont proposé des perspectives intéressantes pour la poursuite de ce travail.

Références

[1] Christian Bessiere, George Katsirelos, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Decomposition of the nvalue constraint. In *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010*, pages 114–128, 2010.

[2] Edgar Frank Codd. Further normalization of the data base relational model. *IBM Research Report, San Jose, California*, RJ909, 1971.

[3] David A. Cohen. A New Classs of Binary CSPs for which Arc-Constistency Is a Decision Procedure. In *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Proceedings*, pages 807–811, 2003.

[4] David A. Cohen, Peter Jeavons, and Marc Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *J. Comput. Syst. Sci.*, 74(5) :721–743, 2008.

[5] M. Cooper, Peter Jeavons, and Andras Salamon. Generalizing constraint satisfaction on trees : hybrid tractability and variable elimination. *Artificial Intelligence*, 174 :570–584, 2010.

[6] Martin C. Cooper, Achref El Mouelhi, Cyril Terrioux, and Bruno Zanuttini. On broken triangles. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014*, pages 9–24, 2014.

[7] R. Dechter. Constraint Networks. In *Encyclopedia of Artificial Intelligence*, volume 1, pages 276–285. John Wiley & Sons, Inc., second edition, 1992.

[8] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34 :1–38, 1987.

[9] Rina Dechter. On the expressiveness of networks with hidden variables. In *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, 2 Volumes.*, pages 556–562, 1990.

[10] Achref El Mouelhi, Philippe Jégou, and Cyril Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, 2013*, pages 947–954, 2013.

[11] Achref El Mouelhi, Philippe Jégou, and Cyril Terrioux. Microstructures for csps with constraints of arbitrary arity. In *Proceedings of the Tenth Symposium on Abstraction, Reformulation, and Approximation, SARA 2013*, 2013.

[12] Achref El Mouelhi, Philippe Jégou, and Cyril Terrioux. Hidden Tractable Classes. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014*, 2014.

[13] Ronald Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst.*, 2(3) :262–278, 1977.

[14] Eugene C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *AAAI*, pages 227–233, 1991.

[15] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :343–282, 2000.

[16] M. Gyssens, P. Jeavons, and D. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66 :57–89, 1994.

[17] C. Han and C. Lee. Comments on Mohr and Henderson's path consistency algorithm. *Artificial Intelligence*, 36 :125–130, 1988.

[18] Peter Jeavons, David A. Cohen, and Martin C. Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1-2) :251–265, 1998.

[19] Philippe Jégou. Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In *Proceedings of the 11th National Conference on Artificial Intelligence, AAAI*, pages 731–736, 1993.

[20] Jean-Louis Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence.*, 10(1) :29–127, 1978.

[21] U. Montanari. Networks of Constraints : Fundamental Properties and Applications to Picture Processing. *Artificial Intelligence*, 7 :95–132, 1974.

[22] D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of ECAI*, pages 125–129, 1994.

[23] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226. ACM, 1978.

[24] K. Stergiou and T. Walsh. Encodings of Non-Binary Constraint Satisfaction Problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, AAAI*, pages 163–168, 1999.

[25] Yuanlin Zhang and Roland H. C. Yap. Arc consistency on n-ary monotonic and linear constraints. In *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference*, pages 470–483, 2000.

Raisonnement disjonctif pour la Contrainte `Cumulative` *

Steven Gay Renaud Hartert Pierre Schaus

UCLouvain, ICTEAM
Place Sainte-Barbe 2
1348 Louvain-la-Neuve, Belgium
{prénom.nom}@uclouvain.be

Résumé

L'ordonnancement est un domaine dans lequel la programmation par contraintes a pu rapidement s'illustrer. Particulièrement, la contrainte `cumulative` – chargée d'assurer la cohérence de l'utilisation limitée d'une ressource par un ensemble de tâches – est l'un des composant critique probablement responsable de ce succès. Malheureusement, assurer la cohérence-de-bornes pour cette contrainte est déjà un problème NP-Difficile. Dès lors, de nombreuses relaxations furent proposées afin de réduire les domaines en des temps polynomiaux parmi lesquelles : Time-Tabling, Edge-Finding, Raisonnement Énergétique et Not-First-Not-Last. Petr Vilim a récemment introduit les raisonnements de type Time-Table Edge-Finding qui renforcent le raisonnement Edge-Finding en considérant le profil obligatoire de la ressource. Nous poursuivons l'idée d'exploiter ce même profil afin de détecter des paires de tâches disjointes de façon dynamique durant la recherche. Ce nouveau type de raisonnement – que nous appelons Time-Table Disjunctive Reasoning – n'est dominé par aucun raisonnement actuellement publié. Nous proposons un algorithme simple, ne se basant sur aucune structure de données complexe, qui implémente une procédure de filtrage basée sur ce raisonnement avec une complexité temporelle de $\mathcal{O}(n^2)$ où n correspond au nombre de tâches. Nos résultats sur des instances connues mettent en exergue les apports de notre nouveau propagateur sur certaines instances et son surcoût dérisoire sur les autres instances.

Abstract

Scheduling has been a successful domain of application for constraint programming since its beginnings. The `cumulative` constraint – which enforces the usage of a limited resource by several tasks – is one of the core components that are surely responsible of this success. Unfortunately, ensuring bound-consistency for the `cumulative` constraint is already NP-Hard. Therefore, several relaxations were proposed to reduce domains in polynomial time such as Time-Tabling, Edge-Finding, Energetic Reasoning, and Not-First-Not-Last. Recently, Vilim introduced the Time-Table Edge-Finding reasoning which strengthens Edge-Finding by considering the time-table of the resource. We pursue the idea of exploiting the time-table to detect disjunctive pairs of tasks dynamically during the search. This new type of filtering – which we call *time-table disjunctive reasoning* – is not dominated by existing filtering rules. We propose a simple algorithm that implements this filtering rule with a $\mathcal{O}(n^2)$ time complexity (where n is the number of tasks) without relying on complex data structures. Our results on well known benchmarks highlight that using this new algorithm can substantially improve the solving process for some instances and only adds a marginally low computation overhead for the other ones.

1 Introduction

De nombreux problèmes réels d'ordonnancement requièrent l'utilisation de ressources cumulatives. Une ressource peut être vue comme une abstraction pour n'importe quelle entité renouvelable – telle que des machines, de l'électricité ou encore de la force de travail – utilisée afin d'exécuter des tâches (aussi appelées activités). Bien que de nombreuses tâches peuvent être exécutées en parallèle sur une même ressource, l'utilisation totale de la ressource à un instant donné est limitée par une capacité fixe. Dans cet article, nous nous concentrons sur une seule ressource cumulative ayant pour capacité discrète $C \in \mathbb{N}$ et un ensemble de tâches $\mathcal{T} = \{1, \ldots, n\}$. Chaque tâche i est caracté-

*Traduction de l'article CPAIOR 2015 [7]

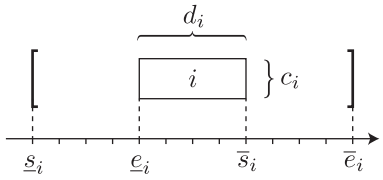


FIGURE 1 – Une tâche i caractérisée par sa date de départ s_i, sa durée d_i, sa fin e_i, et sa consommation de ressource c_i.

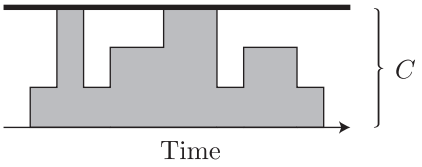


FIGURE 2 – Consommation de ressource cumulée au cours du temps. La contrainte `cumulative` s'assure que la capacité maximale C n'est pas dépassée.

risée par un temps de départ $s_i \in \mathbb{Z}$, une durée fixe $d_i \in \mathbb{Z}$ et un temps de fin $e_i \in \mathbb{Z}$ de manière à ce que l'égalité $s_i + d_i = e_i$ soit vérifiée. Aussi, chaque tâche i consomme une certaine quantité fixe de ressource $c_i \in \mathbb{N}$ pendant son temps d'exécution. Notons que les tâches ne peuvent pas être morcelées pendant leur temps d'exécution. Par la suite, nous dénotons par $\underline{s}_i$ et $\overline{s}_i$ la date de départ au plus tôt et la date de départ au plus tard et par $\underline{e}_i$ et $\overline{e}_i$ la date de fin au plus tôt et la date de fin au plus tard de la tâche i (voir FIGURE 1). La contrainte `cumulative` [1] assure que l'utilisation totale de la ressource ne dépasse jamais sa capacité C et ce pour tout temps t (voir FIGURE 2) :

$$\forall t \in \mathbb{Z} : \sum_{i \in \mathcal{T} : s_i \leq t < e_i} c_i \leq C. \qquad (1)$$

Malheureusement, assurer ne serait-ce que la cohérence-de-bornes est déjà un problème NP-Difficile [12]. Par conséquent, plusieurs relaxations furent proposées ces deux dernières décennies afin de supprimer les valeurs de départ et de fin incohérentes en des temps polynomiaux. Parmi ces relaxations, la règle de filtrage Time-Tabling fut le sujet de nombreuses recherches par la communauté d'ordonnancement [4, 11, 14]. Un algorithme idempotent fut proposé par Letort [11] qui implémente Time-Tabling avec une complexité temporelle de $\mathcal{O}(n^2)$ et a été appliqué avec succès à des problèmes de plusieurs centaines de milliers de tâches. L'algorithme le plus rapide (non-idempotent) connu pour Time-Tabling fut proposé dans [14] et a une une complexité temporelle de $\mathcal{O}(n \log n)$. Malgré son avantage lors du passage à l'échelle, Time-Tabling souffre d'un pouvoir de filtrage limité. À l'autre extrême, les raisonnements de type Raisonnement Énergétique (ER) [3, 5, 6, 15] atteignent des filtrages forts, mais à un coût prohibitif d'une complexité temporelle de $\mathcal{O}(n^3)$. Entre ces deux extrêmes, plusieurs compromis furent proposés afin de trouver un juste équilibre entre pouvoir de filtrage et vitesse d'exécution, e.g., Edge-Finding [17, 8], Time-Table Edge-Finding [18], Time-Table Extended Edge-Finding [14] ou Not-First-Not-Last [16]. Les règles de filtrages ci-dessus sont dominées par le filtrage obtenu par le Raisonnement Énergétique, à l'exception du filtrage obtenu par Not-First-Not-Last, qui n'est par comparable à celui du Raisonnement Énergétique.

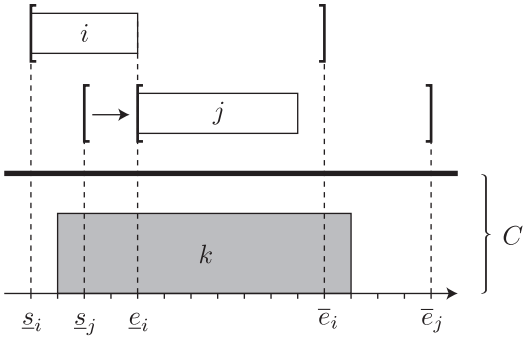


FIGURE 3 – Les tâches i et j ne peuvent pas se superposer à cause de la quantité de ressource déjà consommée par k. Comme j ne peut être assignée avant i, j doit être assignée après i : $e_i < s_j$. Cette situation ne serait pas détectée par l'approche proposée dans [2].

Étonnamment, le filtrage de type Disjunctive Reasoning (DR) [3] n'a été que partiellement adapté au contexte des ressources cumulatives. Dans [2] Baptiste et Le Pape proposent de détecter des ensembles de tâches qui ne peuvent être exécutées en parallèle sans excéder la quantité de ressource disponible initialement. Cependant, cette approche est limitée puisqu'elle ne prend pas en compte les changements dans l'utilisation de la ressource. Cette situation est illustrée dans la FIGURE 3, où une tâche k a été fixée (par la recherche ou par la propagation). Il est facile de voir que les tâches i et j ne peuvent se superposer à cause de la quantité de ressource consommée par la tâche k. Malheureusement, cette situation n'est pas détectée par l'approche proposée par Baptiste et Le Pape.

Dans le travail qui suit, nous proposons d'améliorer Disjunctive Reasoning en considérant les changements de quantité de ressource disponible au cours du temps. De même que dans les travaux de Vilim [18], nous exploitons le profil obligatoire de la ressource – un composant clef de Time-Tabling – pour détecter les paires d'activités disjointes dynamiquement. Notre nouvelle règle de filtrage – que nous appelons Time-

Table Disjunctive Reasoning – n'est dominée par aucune règle de filtrage publiée. Nous proposons un algorithme simple, qui ne repose sur aucune structure de données complexe, permettant d'implémenter cette nouvelle règle de filtrage avec une complexité temporelle de $\mathcal{O}(n^2)$. Nous proposons également plusieurs améliorations à apporter à cet algorithme. Nos résultats, évalués sur les instances de PSPLIB [9], mettent en avant que Time-Tabling Disjunctive Reasoning est une règle de filtrage prometteuse pour les contraintes cumulative à l'état-de-l'art. En effet, utiliser notre algorithme améliore de façon conséquente la résolution de certaines instances tout en ayant un surcoût négligeable sur le temps de résolution des autres instances. Cet article est structuré de la façon suivante. La Section 2 décrit Time-Tabling et les prérequis nécessaires à la lecture de ces travaux. La Section 3 est dédiée à la règle de filtrage Time-Table Disjunctive Reasoning et présente les fondements de l'algorithme. Les résultats sont présentés en Section 4. Enfin, cet article conclut sur les travaux à venir et sur plusieurs améliorations possibles.

2 Profil obligatoire

Même les tâches qui ne sont pas encore fixées apportent des informations qui peuvent être utilisées par les règles de filtrages. Par example, il est possible de déterminer un intervalle sur lequel les tâches disposant d'une fenêtre d'exécution étroite consommeront toujours de la ressource. Un tel intervalle est appelé *partie obligatoire*.

Définition 1 (Partie Obligatoire). *Considérons une tâche $i \in \mathcal{T}$. La partie obligatoire de i est l'intervalle de temps $[\overline{s}_i, \underline{e}_i[$. En d'autres mots, la tâche i possède une partie obligatoire si et seulement si son temps de départ maximum est inférieur à son temps de fin minimum.*

Si la tâche i a une partie obligatoire, nous savons que la tâche i va consommer c_i de ressource pendant toute sa partie obligatoire peu importe le moment auquel la tâche sera réellement exécutée (voir FIGURE 4).

Une version minimale de la consommation de la ressource sur le temps peut être facilement obtenue en agrégeant les partie obligatoires de toutes les tâches. [1] Cette agrégation est généralement appelée profil obligatoire de la ressource.

Définition 2 (Profil obligatoire). *Le profil obligatoire d'une ressource $TT_{\mathcal{T}}$ correspond à l'agrégation des parties obligatoires de toutes les tâches contenues dans $\mathcal{T}$.*

1. Notons que la partie obligatoire d'une tâche fixée correspond à cette même tâche.

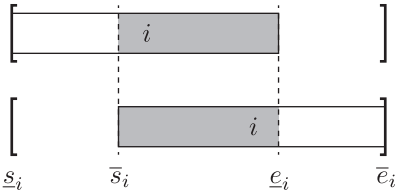


FIGURE 4 – La tâche i a une partie obligatoire $[\overline{s}_i; \underline{e}_i[$ si sa date de début maximale est strictement plus petite que sa date de fin minimale : $\overline{s}_i < \underline{e}_i$. La tâche i consomme nécessairement de la ressource pendant sa partie obligatoire, quel que soit son ordonnancement final.

Cette notion peut être définie formellement comme une fonction escalier :

$$TT_{\mathcal{T}} = t \in \mathbb{Z} \rightarrow \sum_{i \in \mathcal{T} | \overline{s}_i \leq t < \underline{e}_i} c_i. \qquad (2)$$

Le problème est incohérent si $\exists t \in \mathbb{Z} : TT_{\mathcal{T}}(t) > C$.

Le profil obligatoire peut être calculé en $\mathcal{O}(n)$ au moyen d'un algorithme de balayage pour peu que l'on dispose des tâches triées par temps de départ maximum et temps de fin minimum.

3 Time-Table Disjunctive Reasoning

Afin d'expliquer Time-Table Disjunctive Reasoning, nous comptons utiliser quelques notations additionnelles. Pour ce faire, considérons deux intervalles de temps I et J. Nous utilisons le terme J contient I si la relation $I \subseteq J$ est vérifiée. Par ailleurs, nous disons que I et J se superposent si et seulement si $I \cap J \neq \emptyset$.

3.1 Raisonnement disjonctif et intervalle minimal de superposition

Dans [3], Baptiste et al. décrivent brièvement le raisonnement disjonctif dans le contexte cumulatif comme un raisonnement entre toutes les paires de tâches $i \neq j$, qui force la cohérence-de-bornes sur la formule suivante :

$$c_i + c_j \leq C \quad \vee \quad e_i \leq s_j \quad \vee \quad e_j \leq s_i \qquad (3)$$

La règle de filtrage pour mettre à jour le temps de départ d'un tâche peut être trivialement dérivée de cette formule de prédicats.

Proposition 1 (Raisonnement disjonctif). *Considérons une paire de tâches $i \neq j \in \mathcal{T}$ telle que $c_i + c_j > C$, $\underline{s}_j < \underline{e}_i$ et $\overline{s}_i < \underline{e}_j$. Alors, le prédicat $e_i \leq s_j$ doit être vérifié.*

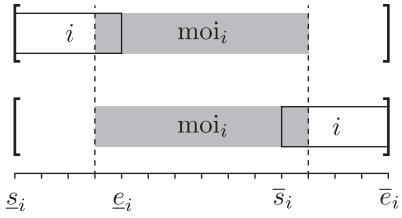


FIGURE 5 – Intervalle minimal de superposition de la tâche i. Quel que soit l'endroit où i est placé, i doit se superposer à moi_i, et moi_i est le plus petit intervalle qui a cette propriété.

Cette règle affirme que si les tâches i et j ne peuvent se superposer et que si j ne peut être ordonnancée au plus tôt sans se superposer à i, alors j ne peut commencer avant la fin minimale de i : $\underline{e_i} \leq \underline{s_j}$. Dans ce contexte, la tâche i est nommée *tâche poussante* alors que la tâche j est appelée *tâche poussée*. Une règle de filtrage pour ajuster la date de fin maximale peut trivialement être dérivée de cette première règle par symétrie.

La règle présentée en PROPOSITION 1 effectue une partie du travail de Time-Tabling. En effet, lorsque $c_i + c_j > C$ et que la tâche i a une partie obligatoire, le raisonnement sur la paire de tâches (i, j) est dominé par celui de Time-Tabling [3]. Cependant, un filtrage supplémentaire peut être dérivé par raisonnement disjonctif lorsque la tâche i n'a pas de partie obligatoire. Cette déduction peut être faite lorsque placer j à son temps de départ minimum causerait une superposition avec i peu importe la position de i. L'idée fondamentale revient à dire que j ne peut s'exécuter sur un intervalle de temps sur lequel i s'exécute au moins à un moment donné. Une contrepartie à la partie obligatoire de i peut donc être définie. Nous la nommons intervalle minimal de superposition de i.

Définition 3. *L'intervalle minimal de superposition d'une tâche i, dénoté moi_i, est l'intervalle de temps le plus petit tel que i s'exécute au moins durant un point de temps de cet intervalle, et ce peu importe le moment auquel la tâche i est exécutée. Formellement, l'intervalle minimal de superposition est défini par $[\underline{e_i} - 1, \overline{s}_i]$.*

L'intervalle minimal de superposition d'une tâche i, moi_i, est illustré en FIGURE 5.

Lorsqu'une tâche possède une partie obligatoire, nous considérons qu'elle ne dispose pas d'intervalle minimal de superposition. En utilisant ce nouveau concept, il est possible de reformuler la partie de la PROPOSITION 1 spécifique au raisonnement disjonctif.

Proposition 2 (Raisonnement disjonctif restreint)**.** *Considérons une paire de tâches $i \neq j$ telle que la tâche i n'a pas de partie obligatoire ($\underline{e_i} \leq \overline{s}_i$) et que $c_i + c_j > C$. Si ordonnancer la tâche j à son temps de départ minimum la fait se superposer complètement à l'intervalle minimal de superposition de i ($\text{moi}_i \subseteq [\underline{s_j}, \underline{e_j}[$), alors le prédicat $\underline{e_i} \leq s_j$ doit être vérifié.*

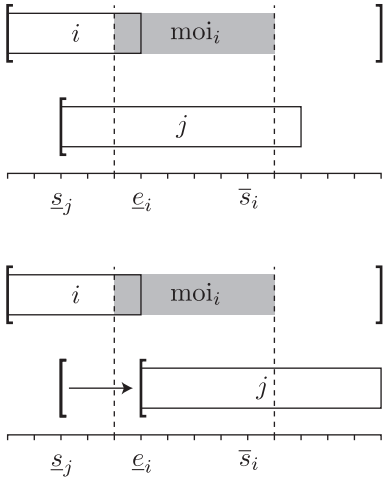


FIGURE 6 – En haut, les tâches i et j ne peuvent pas se superposer pour des raisons de capacité. Assigner s_j à $\underline{s}_j$ forcerait j à contenir moi_i, ce qui rendrait le placement de i impossible. En bas, les valeurs incohérentes de s_j sont filtrées, ce sont les $t \leq \min(\text{moi}_i)$. Ce filtrage est fait en changeant la valeur de $\underline{s}_j$ à $\underline{e}_i$

.

Cette règle présentée en PROPOSITION 2 est illustrée par la FIGURE 6. L'algorithme suivant est directement dérivé de cette règle.

Algorithm 1: Un algorithme appliquant la règle présentée en PROPOSITION 2 en $\mathcal{O}(n^2)$.

Input: un ensemble de tâches $\mathcal{T}$, capacity C
Input: la capacité de la ressource C
Input: un tableau s liant chaque tâche i à $\underline{s}_i$
Output: un tableau s' liant chaque tâche i à son nouveau temps de départ.

```
1 for i ∈ T such that e_i ≤ s̄_i do
2     for j ∈ T − {i} do
3         if c_i + c_j > C ∧ moi_i ⊆ [s_j, e_j[ then
4             s'_j ← max(s'_i, e_i)
```

3.2 Time-Table Disjunctive Reasoning restreint

L'une des faiblesses du raisonnement disjonctif réside dans son incapacité à prendre en compte les changements dans la quantité de ressource disponible avec

le temps (voir FIGURE 3). Dans cette section, nous montrons comment exploiter l'information contenue dans le profil obligatoire (voir Section 2) pour proposer une version renforcée du raisonnement disjonctif que nous appelons *Time-Table Disjunctive Reasoning*.[2]

Nous commençons par introduire Time-Table Disjunctive Reasoning pour un cas simple où les paires de tâches considérées ne possèdent aucune partie obligatoire et ne contribuent donc pas au profil obligatoire. Ce cas particulier nous évite de devoir supprimer les contributions de chaque tâche du profil lorsque nous appliquons le raisonnement disjonctif. Cette restriction sera levée dans la Section 3.3 afin de considérer n'importe quelle paire de tâches.

Considérons donc une paire de tâches $i \neq j$ telle que ni i ni j ne possède de partie obligatoire et que $\text{moi}_i \subseteq [\underline{s}_j, \underline{e}_j[$. Alors, PROPOSITION 2 ne fait que comparer $c_i + c_j$ à C. Cependant, les tâches contenues dans $\mathcal{T} - \{i, j\}$ pourraient ne pas laisser C unités de ressources disponibles au moment où i et j se superposent. Nous pouvons en dériver une nouvelle règle qui prend les parties obligatoires des tâches contenues dans $\mathcal{T} - \{i, j\}$ en compte.

Proposition 3. *Considérons une paire de tâches $i \neq j \in \mathcal{T}$ telle que ni i ni j ne possède de partie obligatoire et que $c_i + c_j + \min_{t \in \text{moi}_i} TT_{\mathcal{T}}(t) > C$. Si exécuter la tâche j à son temps de départ minimum la fait contenir l'intervalle minimal de superposition de i ($\text{moi}_i \subseteq [\underline{s}_j, \underline{e}_j[$), alors le prédicat $\underline{e}_i \leq \overline{s}_j$ doit être vérifié.*

Démonstration. Si la tâche j contient moi_i, alors j devrait augmenter la consommation du profil obligatoire de c_j unité pendant tout moi_i (la tâche j ne possède pas de partie obligatoire et ne contribue donc pas au profil obligatoire). Alors, placer la tâche i n'importe où devrait également augmenter la consommation de la ressource de c_i unités pendant au moins un point de temps $t \in \text{moi}_i$, rendant ainsi la consommation totale au point t supérieure à la capacité de la ressource C. Par ailleurs, puisque $\text{moi}_i \subseteq [\underline{s}_j, \underline{e}_j[$, la durée de j est telle que la placer avant $\underline{e}_i$ la ferait contenir moi_i. Par conséquent, ces valeurs sont incohérentes et le prédicat $\underline{e}_i \leq \overline{s}_j$ doit être vérifié. □

3.3 Time-table Disjunctive Reasoning

La règle de filtrage présentée en PROPOSITION 2 peut être renforcée en utilisant une idée similaire à celle proposée dans [14, 18]. Pour ce faire, nous divisons chaque tâche en deux parties distinctes : sa partie obligatoire et sa *partie libre*.

2. L'idée d'exploiter le profil obligatoire pour renforcer des règles de filtrage existante a déjà été suivie avec succès dans [14, 18].

Définition 4 (Partie libre). *Considérons une tâche $i \in \mathcal{T}$ telle que i possède une partie obligatoire ($\overline{s}_i < \underline{e}_i$). Sa partie libre, dénotée i_f, correspond à une tâche séparée dont le temps de départ minimum et le temps de fin maximum sont les même que ceux de i : $\underline{s}_{i_f} = \underline{s}_i$ and $\overline{e}_{i_f} = \overline{e}_i$. La durée de i_f est égale à la durée de i moins la taille de sa partie obligatoire : $d_{i_f} = d_i - (\underline{e}_i - \overline{s}_i)$. Clairement, si i n'a pas de partie obligatoire alors $i = i_f$.*

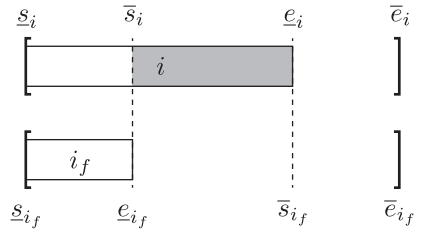


FIGURE 7 – Une tâche i avec sa partie obligatoire et sa partie libre i_f. La partie libre i_f a toujours un intervalle minimal de superposition moi_{i_f}.

Par définition, la partie libre d'une activité n'a pas de partie obligatoire et possède donc toujours un intervalle minimal de superposition (voir FIGURE 7). Par la suite, nous référençons l'ensemble des tâches possédant une partie libre strictement positive par $\mathcal{T}_f = \{i_f \mid i \in \mathcal{T} \wedge d_{i_f} > 0\}$.

Les parties libres nous permettent de généraliser PROPOSITION 2 à toutes les tâches sans devoir se soucier de la présence de parties obligatoires.

Proposition 4 (Time-Table Disjunctive Reasoning). *Considérons une paire de tâches $i_f \neq j_f \in \mathcal{T}_f$ telle que $c_i + c_j + \min_{t \in \text{moi}_{i_f}} TT_{\mathcal{T}}(t) > C$. Si exécuter la tâche j_f à son temps de départ minimum la fait se superposer à l'intervalle minimal de superposition de i_f ($\text{moi}_{i_f} \subseteq [\underline{s}_{j_f}, \underline{e}_{j_f}[$), alors le prédicat $\underline{e}_{i_f} \leq \underline{s}_j$ doit être vérifié.*

Démonstration. Supposons que la partie gauche de l'implication soit vérifiée pour i et j et supposons également que $s_j \leq \min(\text{moi}_i)$. Puisque j_f contient moi_{i_f} lorsque celle-ci est placée à son temps de départ minimum et $d_j \geq d_{j_f}$, placer j avant ou à $\min(\text{moi}_{i_f})$ la fait contenir moi_{i_f}. Observons que j n'a pas de partie obligatoire pendant moi_{i_f}, puisque $\max(\text{moi}_{i_f}) \leq \underline{e}_{j_f} = \min(\underline{e}_j, \overline{s}_j)$.

Si i ne possède pas de partie obligatoire, i ne contribue pas au profil obligatoire. Cela signifie que $\forall t \in \text{moi}_{i_f}, TT(t) = TT_{\mathcal{T}-\{i,j\}}(t)$. Dès lors, placer j avant ou à $\min(\text{moi}_i)$ augmente l'utilisation de la ressource durant moi_{i_f} de c_j unités, ce qui contredit le fait que $i = i_f$ soit placée sur son intervalle minimal de superposition.

Si i possède une partie obligatoire, cette tâche contribue au profil obligatoire exactement durant l'intervalle $[\min(\text{moi}_{i_f}) + 1, \max(\text{moi}_{i_f}) - 1]$. Dès lors, placer j avant ou à $\min(\text{moi}_i)$ augmente l'utilisation de la ressource aux points de temps $\min(\text{moi}_{i_f})$ et $\max(\text{moi}_{i_f})$ de c_j unités. Cela empêche i_f d'être placée vers la gauche ou vers la droite, ce qui par conséquent signifie que la tâche i elle même ne peut pas s'exécuter durant ces points de temps. Cela contredit la définition d'intervalle minimal de superposition puisque i s'exécute pendant au moins un point de temps de cet intervalle. □

L'utilisation des parties libres a deux avantages : (i) Elle permet d'utiliser n'importe quelle tâche i en tant que *tâche poussante*, (ii) Elle permet de ne calculer qu'un fois la hauteur minimale par tâche poussante sans devoir supprimer les parties obligatoires relatives à la contribution de j.

Nous proposons donc un second algorithme capable de tirer avantage des parties obligatoires et libres pour appliquer la règle de filtrage de la PROPOSITION 4 avec une complexité temporelle de $\mathcal{O}(n^2)$. Dans cet algorithme, le profil obligatoire (TT) est encodé dans un simple tableau de paires (t, c) ou t correspond à une date et c à une consommation. Ce tableau est ordonné de façon strictement croissante selon les dates. La primitive $\mathit{consumption}(i, TT)$ (voir ligne 3) peut-être implémentée via la formule $\min_{t \in \text{moi}_{i_f}} TT(t)$, calculable de façon linéaire dans la taille du profil obligatoire. Dès lors, la complexité de cet algorithme est de $\mathcal{O}(n^2)$.

Algorithm 2: Time-Table Disjunctive filtering algorithm.

Input: l'ensemble des tâches $\mathcal{T}$
Input: l'ensemble des tâches poussantes $\mathit{pushing} \subseteq \mathcal{T}_f$
Input: l'ensemble des tâches poussées $\mathit{pushed} \subseteq \mathcal{T}_f$
Input: la capacité de la ressource C
Input: un tableau s' liant chaque tâche i à $\underline{s}_i$
Output: un tableau s' liant chaque tâche i à son nouveau temps de départ.

```
1 TT ← initializeTimeTable(T)
2 for i_f ∈ pushing do
3     gap ← C − c_i − consumption(i, TT)
4     for j_f ∈ pushed − {i_f} do
5         if moi(i) ⊆ [s(j), e(j)[ then
6             if c_j > gap then
7                 s'_j ← max(s'_j, e(i))
```

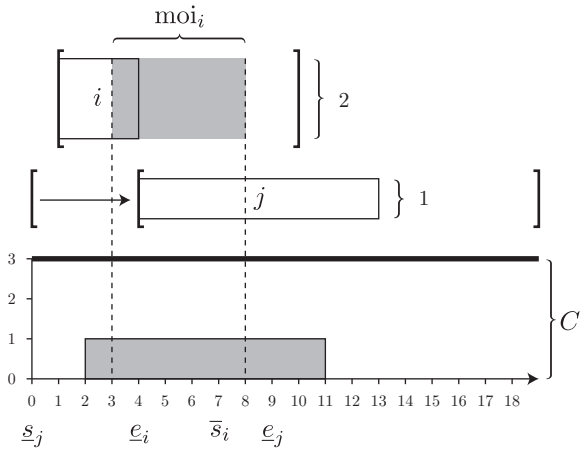


FIGURE 8 – À cause du profil obligatoire, les tâches i et j ne peuvent se superposer. Comme j ne peut être placée avant i, j doit être assignée après i : $e_i \leq s_j$.

Proposition 5. *Le filtrage de TTDR n'est dominé ni par le Raisonnement Énergétique, ni par Not-First-Not-Last.*

Démonstration. FIGURE 8 présente un exemple où TTDR peut filtrer certaines valeurs incohérentes que ni le Raisonnement Énergétique ni Not-First-Not-Last ne sont capables de détecter. Soit deux tâches i et j définie respectivement par $(\underline{s}_i, \overline{e}_i, d_i, c_i) = (2, 11, 3, 2)$, et $(\underline{s}_j, \overline{e}_j, d_j, c_j) = (1, 20, 9, 1)$. Ni la tâche i ni la tâche j ne possède de partie obligatoire i.e. $i_f = i$ et $j_f = j$. Observons également que la condition $\text{moi}_i = [4, 8] \subseteq [\underline{s}_j, \underline{e}_j[= [1, 10[$ est respectée et que le minimum de TT durant $\text{moi}_i = [4, 8]$ est de 1. Cela signifie que les tâches i et j, consommant respectivement 2 et 1 unités de ressource, ne sont pas autorisées à se superposer en $[4, 8]$. Dès lors, $\underline{s}_j$ est mis à jour de sorte que $\min(\text{moi}_i) + 1 = \underline{e}_i = 5$. □

4 Résultats et expériences

Les performances de Time-Table Disjunctive Reasoning (TTDR) furent évaluées sur une célèbre panoplie de tests du Resource Constraints Project Scheduling Problem (RCPSP) de PSPLIB [9]. Le but de ces expériences était de mesurer les performances de plusieurs combinaisons de règles de filtrage de la contrainte cumulative avec et sans TTDR. Voici la liste exhaustive des règles de filtrage considérées :

— Time-Tabling (TT), implémenté au moyen d'une variante rapide de l'algorithme présenté dans [11] ;
— Time-Table Disjunctive Reasoning (TTDR), implémenté selon l'algorithme présenté dans ce papier et les différentes améliorations proposées

dans [7] ;
— Edge-Finding (EF), implémenté comme proposé dans [8] ;
— Raisonnement Énergétique (ER), implémenté avec un algorithme cubique proposé par Baptiste et al. in [3]. Nous avons ajouté quelques améliorations proposées par Derrien et Petit pour réduire le nombre d'intervalles considérés [5] ;
— Not-First-Not-Last (NFNL), implémenté avec un algorithme $\mathcal{O}(n^3)$, une variante de celui proposé par Schutt et al. [16].

Toutes ces règles de filtrage furent implémentées et testées dans le solver open-source OscaR [13]. Nous avons utilisé une recherche SetTimes classique [10], en brisant les égalités par le choix d'une tâche de durée minimale. Ces résultats ont été calculés au moyen d'un processeur 4-core, 8 thread Core(TM) i7-2600 CPU @ 3.40GHz et de 8GB de mémoire RAM, exécuté dans la machine virtuelle Java SE 1.7 JVM.

Nous reportons la distribution cumulée $F(\tau)$ des instances résolues dans les limites calculatoires de τ dans les figures FIGURE 9 et FIGURE 10. Dans la colonne de gauche, τ est le temps, dans celle de droite c'est le nombre de noeuds de l'arbre de recherche ; l'abscisse est logarithmique dans les deux cas $F(\tau) = k$ signifie que k instances ont été résolues en moins de τ ms ou noeuds. Nous avons fixé une limite de 90s pour chaque calcul. [3] A cause du manque de place, nous ne montrons que les résultats obtenus sur les instances à 30 et 120 tâches. Les résultats obtenus avec 60 tâches sont similaires, mais nous avons observé qu'ajouter TTDR avait peu d'effet sur les instances de 90 tâches, où seulement deux instances supplémentaires sont fermées en ajoutant TTDR.

Malgré sa complexité temporelle de $\mathcal{O}(n^2)$, l'algorithme proposé pour TTDR reste très léger. L'unique surcoût de calcul apparait sur de très petits valeurs de temps, lorsque TTDR n'est utilisé qu'en combinaison avec TT. Cependant, le filtrage supplémentaire apporté par TTDR compense rapidement ce léger surcoût et permet de résoudre plus d'une instance pour une limite de temps τ.

Les instances PSPLIB sont connues pour être plus disjonctives que cumulatives. [4] Ajouter des raisonnements à base d'énergie aux instances PSPLIB est un compromis risqué. En effet, les raisonnements à base d'énergie trivialisent peu les instances PSPLIB, alors que TTDR améliore le nombre d'instances résolues par TT seul. Ceci confirme expérimentalement que TTDR

3. C'est pourquoi les graphes de temps n'ont plus de points après 2^{17}ms.

4. Un problème est dit hautement disjonctif lorsque beaucoup de paires de tâches ne peuvent être exécutées en parallèle ; au contraire, un problème est hautement cumulatif si beaucoup de paires de tâches peuvent être exécutées en parallèle [2].

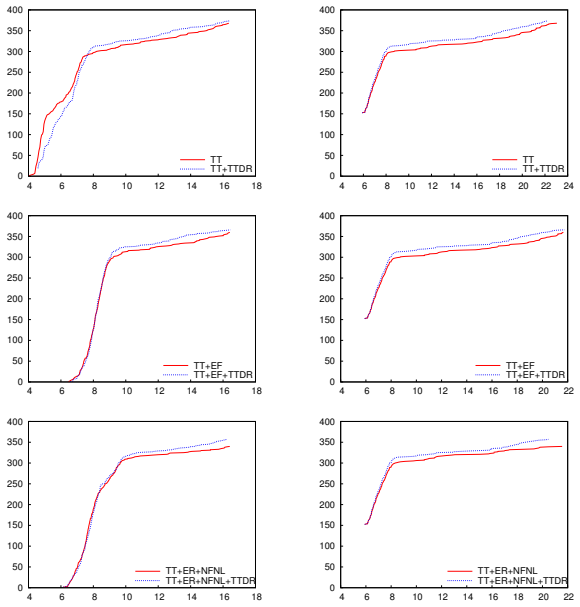


FIGURE 9 – Graphiques pour les instances de PSPLIB30. En ordonnée, le nombre cumulé d'instances résolues. Dans la colonne de gauche, l'abscisse est $log_2(t)$, avec t le temps en ms. Dans la colonne de droite, l'abscisse est $log_2(n)$, avec n le nombre de noeuds pour trouver l'optimal et prouver l'optimalité.

est complémentaire aux filtrages à base d'énergie pour la contrainte `cumulative` (voir Prop. 5)

Finalement, nous voyons aussi que le gain en termes d'instances résolues ne baisse pas quand nous passons des instances à 30 tâches à celles à 120 tâches. Ceci signifie que le filtrage dépend plus de la nature des instances que de leur taille, et que l'algorithme $\mathcal{O}(n^2)$ pour TTDR passe bien à l'échelle.

5 Conclusion

Cet article introduit le Time-Table Disjunctive Reasoning, une nouvelle règle de filtrage qui utilise le profil obligatoire pour détecter des paires disjonctives dynamiquement. En s'appuyant sur les intervalles minimaux de superposition, ce raisonnement accomplit un nouveau type de filtrage qui n'est dominé par aucune règle de filtrage existante telle que le Raisonnement Énergétique ou Not-First-Not-Last. En plus d'être nouvelle, TTDR peut être implémentée avec un simple algorithme $\mathcal{O}(n^2)$ qui ne repose sur aucune structure de données complexe. Les bénéfices de l'utilisation de TTDR en complément à d'autres règles de filtrage ont été évalués sur les instances classiques de PSPLIB. Nos résultats montrent que TTDR est une règle de filtrage prometteuse pour étendre les implémenta-

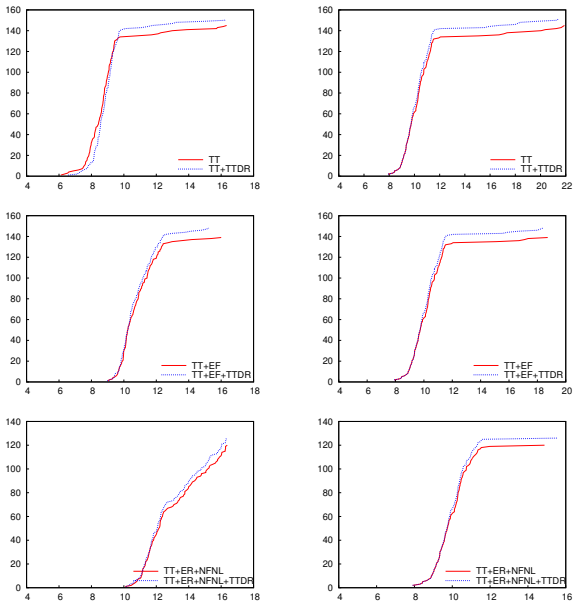


FIGURE 10 – Graphiques pour les instances de PSPLIB120. En ordonnée, le nombre cumulé d'instances résolues. Dans la colonne de gauche, l'abscisse est $log_2(t)$, avec t le temps en ms. Dans la colonne de droite, l'abscisse est $log_2(n)$, avec n le nombre de noeuds pour trouver l'optimal et prouver l'optimalité.

tions état de l'art de la contrainte `cumulative`. En effet, utiliser notre algorithme peut améliorer le processus de résolution de certaines instances substantiellement, et n'ajoute qu'un surcoût dérisoire aux autres.

Bien que le renforcement proposé par TTDR soit déjà un bon compromis en termes de vitesse/filtrage, des améliorations peuvent être envisagées. Par exemple, sa complexité en pratique pourrait être réduite en utilisant des techniques de balayage pour éviter l'examen de paires qui ne se superposent pas. Une amélioration encore plus intéressante concernerait le filtrage : on pourrait le rendre encore plus fort en considérant des intervalles minimaux de superposition de plus d'une tâche à la fois.

Remerciements. Steven Gay est financé par le projet Innoviris 13-R-50 de la région Bruxelles-Capitale. Renaud Hartert est un aspirant du FRS-FNRS. Les auteurs voudraient remercier Sascha Van Cauwelaert pour avoir partagé ses instances et son parseur.

Références

[1] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7) :57–73, 1993.

[2] Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1-2) :119–139, 2000.

[3] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling : applying constraint programming to scheduling problems*, volume 39. Springer, 2001.

[4] Nicolas Beldiceanu and Mats Carlsson. A new multi-resource cumulatives constraint with negative heights. In *CP*, Lecture Notes in Computer Science, pages 63–79, 2002.

[5] Alban Derrien and Thierry Petit. A new characterization of relevant intervals for energetic reasoning. In *Principles and Practice of Constraint Programming*, pages 289–297. Springer, 2014.

[6] Jacques Erschler, Pierre Lopez, and Catherine Thuriot. Scheduling under time and resource constraints. In *Proc. of Workshop on Manufacturing Scheduling, 11th IJCAI, Detroit, USA*, 1989.

[7] Steven Gay, Renaud Hartert, and Pierre Schaus. Time-table disjunctive reasoning for the cumulative constraint. In *CPAIOR*. Springer, 2015.

[8] Roger Kameugne, Laure Pauline Fotso, Joseph Scott, and Youcheu Ngo-Kateu. A quadratic edge-finding filtering algorithm for cumulative resource constraints. *Constraints*, 19(3) :243–269, 2014.

[9] Rainer Kolisch, Christoph Schwindt, and Arno Sprecher. Benchmark instances for project scheduling problems. In *Project Scheduling*, pages 197–212. Springer, 1999.

[10] Claude Le Pape, Philippe Couronné, Didier Vergamini, and Vincent Gosselin. Time-versus-capacity compromises in project scheduling, 1994.

[11] Arnaud Letort, Nicolas Beldiceanu, and Mats Carlsson. A scalable sweep algorithm for the cumulative constraint. In *Principles and Practice of Constraint Programming*, pages 439–454. Springer, 2012.

[12] Wilhelmus Petronella Maria Nuijten. *Time and resource constrained scheduling : a constraint satisfaction approach*. PhD thesis, Technische Universiteit Eindhoven, 1994.

[13] OscaR Team. OscaR : Scala in OR, 2012. Available from `bitbucket.org/oscarlib/oscar`.

[14] Pierre Ouellet and Claude-Guy Quimper. Time-table extended-edge-finding for the cumulative

constraint. In *Principles and Practice of Constraint Programming*, pages 562–577. Springer, 2013.

[15] Eric Pinson. *Le problème de job-shop*. PhD thesis, Paris 6, 1988.

[16] Andreas Schutt and Armin Wolf. A new O(n^2 log n) not-first/not-last pruning algorithm for cumulative resource constraints. In *Principles and Practice of Constraint Programming–CP 2010*, pages 445–459. Springer, 2010.

[17] Petr Vilím. Edge finding filtering algorithm for discrete cumulative resources in O(kn log n). In *Principles and Practice of Constraint Programming-CP 2009*, pages 802–816. Springer, 2009.

[18] Petr Vilím. Timetable edge finding filtering algorithm for discrete cumulative resources. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 230–245. Springer, 2011.

Une contrainte globale pour le lot sizing

Grigori German[1] **Hadrien Cambazard**[1] **Jean-Philippe Gayon**[1] **Bernard Penz**[1]

[1] Univ. Grenoble Alpes, G-SCOP, F-38000 Grenoble, France
CNRS, G-SCOP, F-38000 Grenoble, France
{prénom.nom}@grenoble-inp.fr

Résumé

Cet article présente les premiers éléments d'une approche par contraintes pour la résolution de problèmes de planification de type *lot sizing*. Un problème de *lot sizing* avec bornes inférieures et supérieures de production et de stockage est considéré, et abordé comme une contrainte globale. Cette dernière permet de calculer des bornes inférieures sur le coût global des solutions, mais aussi sur les coûts de setup, les coûts unitaires de production et les coûts de stockage et assure un filtrage efficace. Les tests numériques valident l'approche par rapport aux approches classiques en PLNE, ce qui permet d'envisager son utilisation dans la résolution de problèmes multi-produits difficiles. Mots-clés : *lot sizing*, contrainte globale, programmation par contraintes.

Abstract

This paper presents a constraint programming approach aiming at solving productions planning problems such as *lot sizing* problems. A *lot sizing* problem with lower- and upper bounds of production and storage is studied and considered as a global constraint. This constraint is able to compute cost-related lower bounds - specifically on the global cost, the setup cost, the variable production cost and the storage cost. Numerical tests show the competitiveness of the approach in comparison to classical linear programming approaches and allows us to consider its exploitation in the resolution of hard multi-item problems. Keywords : *lot sizing*, global constraint, constraints programming.

1 Introduction

La planification de production est un domaine riche en problèmes complexes de la recherche opérationnelle et de l'optimisation combinatoire. En particulier, les problèmes de *lot sizing*, introduit par [18], ont été largement étudiés. Beaucoup de ces problèmes sont polynômiaux et abordés par programmation dynamique. En présence de contraintes additionnelles, comme des capacités de production variables au cours du temps, ces problèmes deviennent NP-complets. Ces derniers sont souvent traités par programmation linéaire en nombres entiers. De nombreuses formulations linéaires ont été proposées [13, 3]. Plus récemment, des algorithmes d'approximation ont été proposés pour des variantes NP-difficiles du problème [15]. Très récemment, des travaux se sont intéressés à la résolution de problèmes de *lot sizing* par la programmation par contraintes [8]. Les auteurs introduisent une contrainte globale en planification de production qui considère un ensemble de produits à réaliser avant leurs dates limites de production sur une machine de capacité donnée avec un coût de stockage limité. L'approche traitée dans [8] relève cependant davantage de l'ordonnancement (détermination des dates de début des activités) alors que le présent papier considère un horizon temporel plus tactique (détermination des quantités à produire sur un horizon discrétisé), ce qui différencie les problèmes considérés.

L'objectif de cet article est d'introduire une contrainte globale pour un problème de *lot sizing* mono-produit relativement général, prenant en compte des capacités et des coûts variables au cours de l'horizon de planification. On considère à la fois des coûts de setup et les coûts unitaires (par unité de produit) de production, des coûts unitaires de stockage, ainsi que des capacités de production et de stockage. Nous pensons que cette contrainte globale est une brique importante pour la modélisation et la résolution de nombreux problèmes multi-produits difficiles. Nous posons les bases algorithmiques pour la réalisation du filtrage. Des résultats expérimentaux préliminaires valident les bornes inférieures proposées en comparant l'approche avec des modèles de Programmation Linéaire en Nombres Entiers (PLNE). Le filtrage apporte

des informations intéressantes sur les domaines des variables de décision (production et stockage) ainsi que des bornes inférieures des coûts pris individuellement (coûts de setup de production, coûts variables de production, coûts de stockage).
L'article est structuré de la façon suivante : Une définition de la contrainte globale est proposée dans la section 3. Des algorithmes de filtrage sont ensuite présentés dans les sections 4 et 5. La description des expérimentations ainsi que l'analyse des premiers résultats obtenus font l'objet de la section 6. La section 7 illustre l'utilisation de la contrainte globale proposée dans cet article au sein de différents problèmes déjà abordés dans la littérature. Pour finir des pistes de recherche sont proposées dans la section 8.

2 Rappel sur la programmation par contraintes et notations générales

Un problème de satisfaction de contraintes $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, est constitué d'un ensemble de variables $\mathcal{X}$, un ensemble de domaines $\mathcal{D}$ spécifiant un domaine (ensemble fini de valeurs) pour chaque variable et un ensemble $\mathcal{C}$ de contraintes. Une contrainte détermine les combinaisons autorisées de valeurs pour un sous-ensemble de variables. Les variables, *e.g* X_i, sont notées par des lettres majuscules. $\overline{X_i}$ et $\underline{X_i}$ dénotent respectivement les bornes supérieure et inférieure du domaine de X_i tandis que $D(X_i)$ désigne son domaine. Étant donnée une contrainte C sur un ensemble de variables $\langle X_1, \ldots, X_n \rangle$, un *support de bornes* pour C est un tuple $\langle v_1, \ldots, v_n \rangle$ de valeurs qui soit solution de / qui vérifie C et tel que $\underline{X_i} \leq v_i \leq \overline{X_i}$ pour toute variable X_i. Une variable X_i de C est dite cohérente aux bornes (BC) pour C si $\underline{X_i}$ et $\overline{X_i}$ appartiennent à un support de bornes de C. La contrainte C est dite BC si toutes ses variables sont BC.

3 Une contrainte globale pour le *lot sizing*

3.1 Présentation du problème et des notations

Le problème de *lot sizing* consiste à planifier la production d'un unique type de produit sur un horizon fini de T périodes afin de satisfaire une demande d_t à chaque période t. Le coût de production à une période t est constitué d'un coût unitaire p_t (coût payé par produit) et d'un coût de setup s_t payé si au moins une unité est produite. Le coût de stockage est un coût unitaire h_t payé par produit en stock à la fin de la période t. De plus, la production (respectivement le stockage) est limitée par des capacités minimales et maximales $\underline{\alpha_t}$ et $\overline{\alpha_t}$ (respectivement $\underline{\beta_t}$ et $\overline{\beta_t}$) à chaque période. L'objectif est de définir un plan de production satisfaisant les demandes à chaque période, respectant les capacités et minimisant le coût global du plan de production. Les notations utilisées pour la représentation des données et pour les variables sont résumées ci-dessous :

Paramètres :
- T : Nombre de périodes.
- $p_t \in \mathbb{R}_+$: Coût unitaire de production à t.
- $h_t \in \mathbb{R}_+$: Coût unitaire de stockage entre les périodes t et $t+1$.
- $s_t \in \mathbb{R}_+$: Coût de setup à t (payé si au moins une unité a été produite à t).
- $d_t \in \mathbb{N}$: Demande à la période t.
- $\underline{\alpha_t}, \overline{\alpha_t} \in \mathbb{N}$: Quantités minimales et maximales de production à la période t.
- $\underline{\beta_t}, \overline{\beta_t} \in \mathbb{N}$: Quantités minimales et maximales de stockage à la période t.
- $I_0 \in \mathbb{N}$: Niveau de stock initial.

Variables :
- $X_t \in \mathbb{N}$: Quantité produite à t.
- $I_t \in \mathbb{N}$: Quantité stockée entre t et $t+1$.
- $Y_t \in \{0,1\}$: Vaut 1 si au moins une unité est produite à t, 0 sinon.
- $C \in \mathbb{R}_+$: Coût global du problème.
- $Cp \in \mathbb{R}_+$: Somme des coûts unitaires de production.
- $Ch \in \mathbb{R}_+$: Somme des coûts unitaires de stockage.
- $Cs \in \mathbb{R}_+$: Somme des coûts de setup de production.

La figure 1 schématise le problème, et un modèle mathématique pour ce problème (L) peut s'écrire :

$$\text{Minimize } C = Cp + Ch + Cs$$

$$
\begin{array}{lll}
\text{(L.1)} & I_{t-1} + X_t - I_t = d_t & \forall\, t = 1 \ldots T \\
\text{(L.2)} & \underline{\alpha_t} \leq X_t \leq \overline{\alpha_t} & \forall\, t = 1 \ldots T \\
\text{(L.3)} & \underline{\beta_t} \leq I_t \leq \overline{\beta_t} & \forall\, t = 1 \ldots T \\
\text{(L.4)} & X_t \leq \overline{\alpha_t} Y_t & \forall\, t = 1 \ldots T \\
\text{(L.5)} & Cp = \sum_{t=1}^{T} p_t X_t & \\
\text{(L.6)} & Ch = \sum_{t=1}^{T} h_t I_t & \\
\text{(L.7)} & Cs = \sum_{t=1}^{T} s_t Y_t & \\
\text{(L.8)} & X_t, I_t \in \mathbb{N}, Y_t \in \{0,1\} & \forall\, t = 1 \ldots T
\end{array}
\quad (L)
$$

Les contraintes $(L.1)$ sont les contraintes de conservation du flot pour chaque période, les contraintes $(L.2)$ et $(L.3)$ sont les contraintes de capacités, les contraintes $(L.4)$ sont les contraintes de setup ; notons également que le paiement d'un coût de setup à t (*i.e* $Y_t = 1$) n'implique pas forcément une production à t (*i.e* $X_t > 0$), ce qui sera utile dans la modélisation

de problèmes multi-produits pour lesquels le coût de setup est partagé. Finalement, les contraintes ($L.5$), ($L.6$) et ($L.7$) sont les expressions des différents coûts.

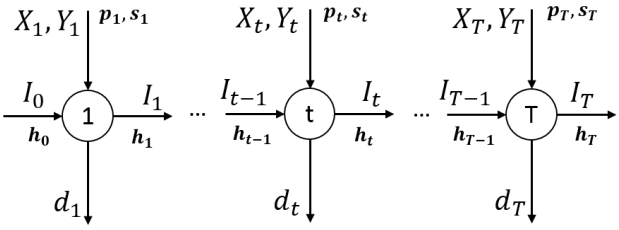


FIGURE 1 – Lot sizing

Notons que cette formulation est plus générale que la présentation classique du *lot sizing* car elle inclut des quantités minimales à produire et à stocker.

Des cas particuliers du problème (L) on été largement étudiés dans la littérature suivant les différents paramètres pris en compte. En 1958, Wagner et Whitin introduisent le problème fondateur dans [18], sans capacités, et le résolvent à l'aide d'un programme dynamique en $O(T^2)$. Cette complexité a depuis été améliorée en $O(T \log T)$ dans [5, 17, 1]. Le problème avec capacités de production constantes et coûts concaves a été résolu en $O(T^4)$ dans [6]. Avec des hypothèses de coûts linéaires pour le stockage, la complexité a pu être abaissée à $O(T^3)$ dans [16]. Le *lot sizing* avec capacités de stockage mais sans coûts de setup est résolu dans [12] en $O(T^3)$ puis résolu avec capacités de stockage et coûts de setup en $O(T^2)$ dans [2].

Les problèmes de *lot sizing* mono-produit avec des coûts non concaves ou capacités de production variables au cours du temps sont eux NP-difficiles [4].

3.2 Sémantique de la contrainte globale

Dans cette section, nous définissons formellement la contrainte globale. La contrainte porte sur les vecteurs de variables $X = \langle X_1, \ldots, X_T \rangle$, $I = \langle I_1, \ldots, I_T \rangle$ et $Y = \langle Y_1, \ldots, Y_T \rangle$ et les quatre variables de coûts Cs, Cp, Ch et C, ces variables correspondant aux variables de (L). Enfin l'ensemble des paramètres du problème est noté $data = \{I_0\} \cup \{(p_t, h_t, s_t, d_t) | t \in \{1, \ldots, T\}\}$.

Définition 1 LOTSIZING$(X, I, Y, Cp, Ch, Cs, C, data)$ *possède une solution si et seulement si il existe un plan de production solution de (L) ayant des coûts de production et stockage inférieurs ou égaux à* $\overline{Cp}$, $\overline{Cs}$, $\overline{Ch}$ *et un coût global inférieur ou égal à* $\overline{C}$.

Propriété 1 *Réaliser la consistance de bornes (BC) pour la contrainte de* LOTSIZING *est NP-Difficile.*

Preuve : La contrainte LOTSIZING possède un *support de bornes* si et seulement si il existe un plan de production respectant les capacités de production ($\underline{X_t}$, $\overline{X_t}$), les capacités de stockage ($\underline{I_t}$, $\overline{I_t}$) et les bornes supérieures des coûts. Obtenir un plan de production de coût inférieur à $\overline{C}$ seul est un problème de *lot sizing mono-produit avec capacités* qui est NP-difficile [4]. Vérifier si une borne appartient à un support de bornes est donc NP-difficile et la réalisation de la consistance de bornes également. □

4 Réalisabilité et élimination des bornes inférieures de production/stockage

Les relaxations proposées dans la suite de l'article n'ont à notre connaissance pas été traitées auparavant sur ce type de problème, en particulier l'utilisation du Weighted Interval Scheduling Problem présentée à la section 5.2.
On ignore dans un premier temps le respect des bornes supérieures des coûts : l'algorithme 1 effectue la consistance de bornes des équations (L.1) $I_{t-1} + X_t - I_t = d_t$ et détecte si le problème est réalisable du point de vue des capacités et de la demande.

Algorithm 1 testerRealisabilite

```
1: boolean modif ← true
2: while modif do
3:     modif ← false
4:     for t = 1...T do
5:         modif ← modif ∨ majBornes(t)
6: end algorithm
```

L'algorithme $majBornes(t)$ met à jour les bornes des variables X_t et I_t pour une période donnée et renvoie un booléen qui indique si une des bornes a été modifiée. Les mises à jour à t sont les suivantes :

- $\underline{X_t} \geq d_t - \overline{I_{t-1}} + \underline{I_t}$
- $\overline{X_t} \leq d_t - \underline{I_{t-1}} + \overline{I_t}$
- $\underline{I_t} \geq -d_t + \underline{I_{t-1}} + \underline{X_t}$ et $\underline{I_t} \geq d_{t+1} + \underline{I_{t+1}} - \overline{X_{t+1}}$
- $\overline{I_t} \leq -d_t + \overline{I_{t-1}} + \overline{X_t}$ et $\overline{I_t} \leq d_{t+1} + \overline{I_{t+1}} - \underline{X_{t+1}}$

Propriété 2 *Il y a équivalence entre :*

1. LOTSIZING *possède un plan de production respectant les capacités de stockage et de production.*
2. $\forall t \in \{1..T\}, D(X_t) \neq \emptyset$ ***et*** $D(I_t) \neq \emptyset$ *après l'exécution de l'algorithme 1.*

Preuve :
Cas 1 ⇒ 2 : direct.
Cas 2 ⇒ 1 : Supposons que les bornes et variables X_t et I_t ont été mises à jour par l'algorithme 1. Construisons par récurrence une solution réalisable. On pose

Algorithm 2 ecouler

Entrée : Q, t

1: $i \leftarrow t$
2: **while** $Q > 0$ **do**
3: $\quad d'_i \leftarrow \max\{0, d_i - Q\}$
4: $\quad Q \leftarrow Q - d_i$
5: $\quad$ **if** $Q > 0$ **then**
6: $\quad\quad \overline{I'_i} \leftarrow \overline{I_i} - Q$
7: $\quad\quad \underline{I'_i} \leftarrow \max\{0, \underline{I_i} - Q\}$
8: $\quad i \leftarrow i + 1$
9: **end algorithm**

Algorithm 3 transformerProbleme

1: **for** $t = 1 \dots T$ **do**
2: $\quad$ **if** $\underline{Y_t} > 0$ **then**
3: $\quad\quad s'_t \leftarrow 0$
4: $\quad$ **if** $\underline{X_t} > 0$ **then**
5: $\quad\quad \overline{X'_t} \leftarrow \overline{X_t} - \underline{X_t}$
6: $\quad\quad \underline{X'_t} \leftarrow 0$
7: $\quad\quad ecouler(\underline{X_t}, t)$
8: $\quad$ **if** $\underline{I_t} > 0$ **then**
9: $\quad\quad d'_t \leftarrow d_t + \underline{I_t}$
10: $\quad\quad \overline{I'_t} \leftarrow \overline{I_t} - \underline{I_t}$
11: $\quad\quad \underline{I'_t} \leftarrow 0$
12: $\quad\quad ecouler(\underline{I_t}, t+1)$
13: **end algorithm**

$X_1 = \underline{X_1}$ et $I_1 = \underline{I_1}$ qui vérifient la conservation du flot (d_1 est satisfaite) et respecte les capacités à $t = 1$. Supposons qu'on dispose d'une solution réalisable pour les périodes $1, \dots, t-1$ telle que $I_{t-1} = \underline{I_{t-1}}$. On obtient une solution réalisable pour t en posant :

(a) $X_t = d_t + \underline{I_t} - \underline{I_{t-1}}$ et (b) $I_t = \underline{I_t}$

On vérifie la conservation du flot à t (satisfaction de la demande assurée par (a)) et si X_t ne viole pas la capacité de production. Or d'après l'algorithme 1 :

$\underline{I_t} \geq -d_t + \underline{I_{t-1}} + \underline{X_t}$ et $\underline{I_{t-1}} \geq d_t + \underline{I_t} - \overline{X_t}$

En remplaçant $\underline{I_t}$ dans (a) on vérifie que $X_t \geq \underline{X_t}$ et en remplaçant $\underline{I_{t-1}}$ on vérifie $X_t \leq \overline{X_t}$. □

Dans la suite, on suppose que les contraintes de conservation du flot (L.1) ont été propagées jusqu'au point fixe (consistance de bornes) ainsi que les implications $X_t > 0 \implies Y_t = 1$ (L.4). On note P le problème correspondant.

Afin de simplifier les algorithmes de filtrage, P peut être transformé davantage en supprimant les bornes inférieures sur la production et les stocks. L'idée est que s'il existe t tel que $\underline{X_t} > 0$, on peut écouler $\underline{X_t}$ sur les demandes suivantes - c'est-à-dire que $\underline{X_t}$ satisfait un certain nombre de demandes à partir de t. De même si $\underline{I_t} > 0$, on peut reporter $\underline{I_t}$ sur les périodes suivantes. Cette quantité doit alors être produite à une période précédente. On ignore laquelle mais cela revient à augmenter d_t de $\underline{I_t}$.

Le problème obtenu après transformation est noté P' et ses variables et paramètres sont notés $X', I', Y', Cp', Ch', Cs', C', p', h', s', d'$. L'algorithme 2 permet d'écouler une quantité donnée Q sur les demandes à partir de la période t donnée. L'algorithme 3 permet de transformer P en éliminant les bornes inférieures des variables X_t et I_t. Ainsi on s'assure que P' vérifie : $\underline{X'_t} = 0$, $\underline{I'_t} = 0$ et $Y_t = Y'_t$. La transformation affecte également les coûts de setup et les demandes.

La figure 2 donne un exemple de transformation de P sur un horizon $T = 3$. Les nombres entre crochets représentent le domaine de X_t et I_t. La première étape consiste à éliminer la quantité minimale à produire à la période 1 en l'écoulant sur les demandes suivantes. La deuxième étape permet d'éliminer la quantité minimale à stocker entre les périodes 2 et 3 en l'écoulant sur la demande suivante. On observe bien qu'alors la demande d_2 est augmentée (flèche plus large). La dernière étape permet d'éliminer la borne inférieure de X_3 en diminuant la demande d_3.

Propriété 3 *A toute solution optimale de P correspond une solution optimale de P' de coût égal au coût optimal de P plus une constante.*

Preuve : Écouler $\underline{X_t}$ sur les demandes suivantes permet de prendre en compte un coût de production qui est obligatoire dans P sans affecter les variables de décision du problème. De même, on peut écouler $\underline{I_t}$ sur les demandes suivantes et ainsi comptabiliser un coût de stockage obligatoire. Afin d'éviter de prendre une décision quant au coût de production induit par cette quantité, on la reporte sur la demande d_t, ce qui assure la conservation du flot à t. □

En supprimant les bornes inférieures de production et de stockage, il faut comptabiliser les coûts qu'on a été obligé de payer. Définissons $Cp_{min}, Ch_{min}, Cs_{min}$ et C_{min}, les coûts à comptabiliser après le changement de variable. On rappelle qu'on suppose que la consistance de bornes sur les équations du flot a été atteinte.

$Cp_{min} = \sum_{t=1}^{T} p_t \underline{X_t}$

$Ch_{min} = \sum_{t=1}^{T} h_t \underline{I_t}$

$Cs_{min} = \sum_{t|\underline{Y_t}=1} s_t$

$C_{min} = Cp_{min} + Ch_{min} + Cs_{min}$

Les relations entre les variables de P et P' permettent de propager les bornes obtenues par la suite avec P' dans P :

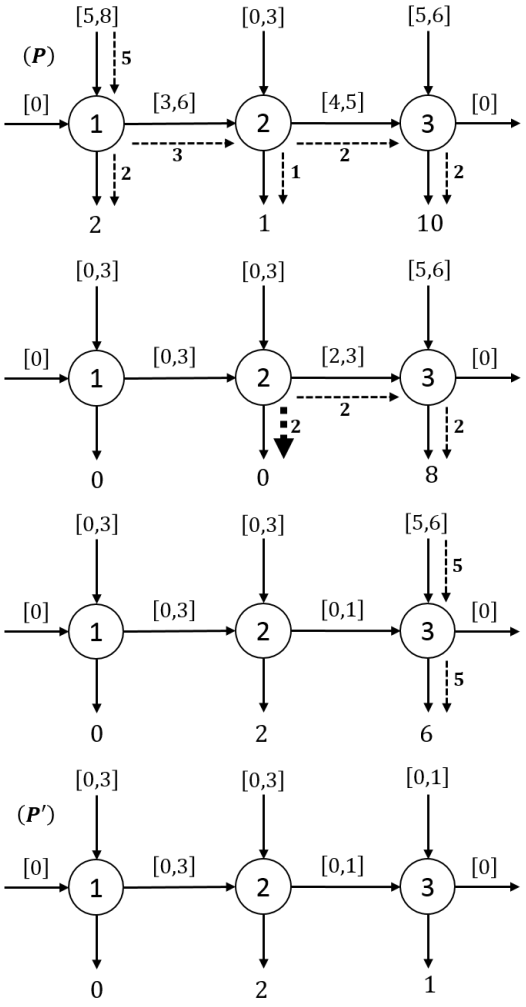


FIGURE 2 – Exemple de transformation avec $T = 3$

$Cp = Cp' + Cp_{min}$ et $Ch = Ch' + Ch_{min}$

$Cs = Cs' + Cs_{min}$ et $C = C' + C_{min}$

$X_t = X'_t + \underline{X_t}$ et $I_t = I'_t + \underline{I_t}$

5 Calcul de bornes inférieures des coûts et filtrage

Le problème P' reste NP-difficile et on étudie différentes relaxations pour calculer des bornes sur les coûts et les variables afin de les propager dans P.

5.1 Bornes sur C'

Une première relaxation porte sur les coûts de setup en les répartissant sur toute la capacité de production à chaque période. On pose $\widetilde{p}_t = p_t + \frac{s'_t}{\overline{X'_t}}$ si $\overline{X'_t} > 0$ et $\widetilde{p}_t = p_t$ sinon. Dans le cas où l'on produit à pleine capacité à t, on va payer $p_t\overline{X'_t} + s'_t$, sinon on ne va payer qu'une fraction du coût de setup. Il s'agit donc bien d'une borne inférieure du coût de production à la période t.

On définit le graphe $G = (V, E)$ (cf. figure 3) tel que :

- V est composé de $T + 2$ nœuds, un nœud t pour chaque période, une source s et un puits p.
- E est composé de $3T - 1$ arêtes. Chaque nœud t possède deux arcs sortants : (t, p) avec un coût nul et une capacité d'_t ainsi que $(t, t + 1)$ (sauf pour T) avec un coût h_t et une capacité $\overline{I'_t}$. Enfin T arcs (s, t) partent de la source vers chaque nœud t avec un coût $\widetilde{p}_t$ et une capacité $\overline{X'_t}$.

Notons $C_{flotPHS}$ le coût minimal d'un flot de quantité $\sum_{t=1}^{T} d'_t$ entre s et p.

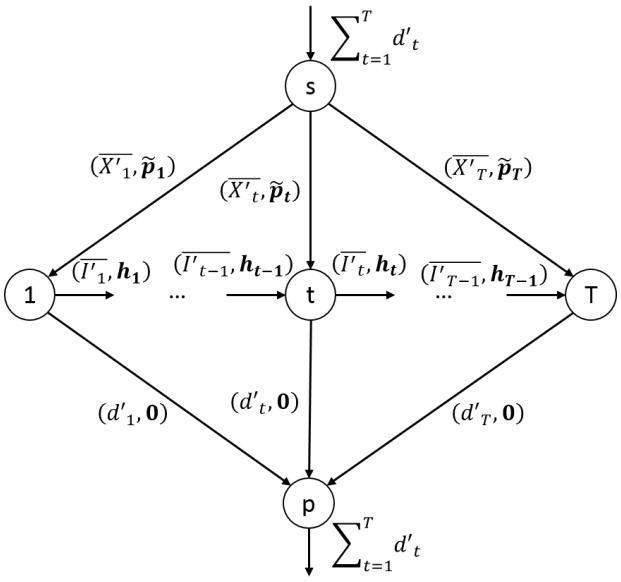


FIGURE 3 – Graphe G pour une borne inférieure de C'

Propriété 4

$$\underline{C'} \geq C_{flotPHS}$$

Preuve : Toute solution de P' est un flot réalisable dans G. □

On peut propager une deuxième borne sur C' en s'appuyant sur $\underline{Cs'}$ et sur une borne prenant en compte uniquement les coûts unitaires de production et de stockage. Cette borne est calculée en faisant circuler $\sum_{t=1}^{T} d'_t$ unités de flot dans un graphe G_2 défini comme G mais avec $\widetilde{p}_t = p_t$. Ce nouveau réseau permet de prendre en compte tous les coûts à l'exception du setup. Soit C_{flotPH} le coût optimal du problème de flot sur G_2. On a alors

$$\underline{C'} \geq C_{flotPH} + \underline{Cs'}$$

5.2 Borne sur Cs'

Le problème est difficile dès que des capacités de production et des coûts de setup sont présents.

Propriété 5 *Le problème* (L) *avec* $h_t = p_t = \underline{\beta_t} = \underline{\alpha_t} = 0$ *et* $\overline{\beta_t} = +\infty$ *est NP-difficile.*

Preuve :

La réduction se fait à partir du problème Sac-à-dos : *Entrée* : n objets de poids w_i, de valeurs v_i. Deux entiers V et W.
Question : Existe-t-il un ensemble S d'objets tel que $\sum_{i\in S} v_i \geq V$ et $\sum_{i\in S} w_i \leq W$?

On pose $s_t = w_t$, $\overline{X_t} = v_t$, $\forall t = 1 \dots T$ et $d = \{0, .., 0, V\}$. On cherche un plan de production tel que $\sum_{t\in S} X_t \geq V$ et $\sum_{t\in S|X_t>0} s_t \leq W$. N'importe quelle solution du *lot sizing* est équivalente à une solution où l'on produit à pleine capacité. On peut donc se restreindre à cet ensemble de solutions. La transformation est polynômiale. Une solution au problème de sac-à-dos est aussi solution du *lot sizing*. Et inversement une solution du problème de *lot sizing* est une solution du sac-à-dos. □

Nous proposons une approche par programmation dynamique pour obtenir une borne inférieure de Cs'. L'idée est d'associer une borne inférieure du coût de setup à chaque sous-plan de production possible et de chercher un ensemble de sous-plans disjoints maximisant la somme de ces valeurs.

On considère un sous-plan de production, c'est-à-dire un intervalle de périodes $[t_1, t_2]$ $(t_1 < t_2)$. On cherche à borner le nombre minimum de périodes de production $n_{t_1t_2}$ dans un tel sous-plan pour obtenir une borne inférieure $w_{t_1t_2}$ du coût de setup nécessaire à l'intervalle $[t_1, t_2]$.

Les périodes entre t_1 et t_2 sont simplement ordonnées par capacités de production décroissantes d'une part et par coûts de setup croissants d'autre part. On considère donc un ordre $\gamma_1, \dots, \gamma_{t_2-t_1+1}$ des périodes de sorte que : $\overline{X'_{\gamma_1}} \geq \dots \geq \overline{X'_{\gamma_{t_2-t_1+1}}}$

Et un ordre $\delta_1, \dots, \delta_{t_2-t_1+1}$ des périodes de sorte que : $s_{\delta_1} \leq \dots \leq s_{\delta_{t_2-t_1+1}}$

Pour estimer la quantité minimale à produire sur $[t_1, t_2]$, on suppose que le stock maximum possible arrive en t_1 et le stock minimum possible est laissé en t_2 *i.e.* que $I'_{t_1-1} = \overline{I'_{t_1-1}}$ et $I'_{t_2} = 0$. De plus, on suppose que la production est maximum aux périodes déjà actives et ainsi couvrir $Q_{t_1,t_2} = \sum_{t\in[t_1,t_2]|Y'_t=1} \overline{X'_t}$ unités. Pour couvrir la demande restante dans $[t_1, t_2]$ il faut donc activer au moins $n_{t_1t_2}$ périodes :

$$n_{t_1t_2} = \min\{k \mid \sum_{j=1}^{k} \overline{X'_{\gamma_j}} \geq \sum_{t=t_1}^{t_2} d_t - \overline{I'_{t_1-1}} - Q_{t_1,t_2}\}$$

Le coût de setup minimum nécessaire au sous-plan $[t_1, t_2]$ est simplement la somme des $n_{t_1t_2}$ plus petits coûts de setup des Y' non instanciés ($|D(Y'_t)| = 2$) soit :

$$w_{t_1t_2} = \sum_{j=1,\dots,n_{t_1t_2}| \, |D(Y'_{\delta_j})|=2} s_{\delta_j}$$

Tous les sous-plans possibles (il y en a donc $\frac{T(T-1)}{2}$) sont ordonnés par dates de fin croissantes d'abord puis dates de début croissantes : $[1,2], [1,3], [2,3], [1,4], [2,4], [3,4], \dots, [T-1, T]$. Pour le $i^{ème}$ sous-plan correspondant à l'intervalle $[t_1, t_2]$ on note w_i la borne inférieure de son coût de setup c'est à dire $w_i = w_{t_1t_2}$. On remarque que n'importe quel ensemble de sous-plans disjoints permet d'obtenir une borne inférieure du coût de setup global en considérant la somme des w_i associés à ces sous-plans disjoints.
Pour obtenir la meilleure borne inférieure selon ce principe, on cherche donc **un ensemble S de sous-plans disjoints qui maximise** $\sum_{k\in S} w_k$. Ce problème est précisément le Weighted Interval Scheduling Problem [9]. Il s'agit d'un problème polynômial résolu dans le cas général en $O(n \log n)$ (avec n le nombre d'intervalles), en s'appuyant sur un tri des intervalles en $O(n \log n)$ et un programme dynamique de complexité linéaire $O(n)$. Dans notre cas, les intervalles sont déjà ordonnés et l'algorithme de programmation dynamique peut donc être appliqué en $O(T^2)$. On considère l'ordre des intervalles indiqué plus haut et on note $f^*(i)$ le poids maximum possible avec les i premiers intervalles. En considérant $f^*(0) = 0$, on peut écrire pour tout $i = 1, \dots, \frac{T(T-1)}{2}$ que :

$$f^*(i) = \max(f^*(i-1), f^*(p_i) + w_i)$$

où p_i est le plus grand entier, plus petit que i (donc $p_i < i$), tel que les intervalles p_i et i sont disjoints. Ainsi p_i est le premier intervalle avant le $i^{ème}$ et compatible temporellement avec le $i^{ème}$. Par exemple $[3, 4]$ est le $6^{ème}$ intervalle et $p_6 = 1$ car $[1, 2]$ est compatible avec $[3, 4]$ mais $[1, 3]$ ne l'est pas. On cherche donc la valeur $C_{dynS} = f^*(\frac{T(T-1)}{2})$.

Propriété 6

$$\underline{Cs'} \geq C_{dynS}$$

Preuve : La validité s'appuie sur deux éléments. w_{t_1,t_2} représente une borne inférieure sur le coût de setup du sous-plan $[t_1, t_2]$. De plus, n'importe quel ensemble de sous-plans disjoints permet d'obtenir une borne inférieure du coût de setup global en considérant la somme des w_i associés à ces sous-plans. □

5.3 Borne sur Ch'

Pour obtenir une borne sur le coût de stockage, on relâche les coûts de setup et les coûts unitaires de pro-

Algorithm 4 produireAuPlusTard

```
1: C_glouH ← 0
2: for t = 1...T do
3:     UBX_t ← X̄_t
4: for t = T...1 do
5:     Q ← d_t
6:     i ← t
7:     while Q > 0 do
8:         if i ≠ t then
9:             C_glouH ← C_glouH + h_t Q
10:        prod ← min{Q, UBX_i}
11:        Q ← Q − prod
12:        UBX_i ← UBX_i − prod
13:        i ← i − 1
14: end algorithm
```

duction (*i.e.* $s'_t = p'_t = 0$). Ce problème relaxé est polynômial et il suffit d'un algorithme qui produit chaque demande au plus tard en respectant les capacités (voir algorithme 4). Cet algorithme est présenté en $O(T^2)$ par souci de clarté mais peut aussi s'écrire en $O(T)$.

Soit C_{glouH} le coût optimal de ce problème relaxé :

$$\underline{Ch'} \geq C_{glouH}$$

5.4 Borne sur Cp'

On définit le graphe G_3 comme G (cf. 5.1) avec $\widetilde{p}_t = p_t$ et $h_t = 0$. Le coût minimum d'un flot de $\sum_{t=1}^{T} d'_t$ unités sur ce graphe donne une borne sur le coût unitaire de production.

Soit C_{flotP} le coût optimal de ce problème de flot. On a alors

$$\underline{Cp'} \geq C_{flotP}$$

5.5 Filtrage par coûts réduits

Rappelons que nous faisons l'hypothèse ici qu'une solution réalisable existe (il existe ainsi un flot réalisable pour le réseau G) et qu'il n'y a pas de bornes inférieures sur les quantités à produire et à stocker.

La résolution du flot de coût minimum fournit à l'optimum des coûts réduits pour les arcs de production *i.e* les arcs (s,t) de G et les arcs de stockage entre deux périodes consécutives *i.e* les arcs $(t,t+1)$. Ces coûts réduits peuvent être utilisés pour filtrer les bornes des variables X'_t et I'_t comme dans [7].
Considérons un arc de stockage $(t,t+1) \in E$. Son coût réduit est défini par :

$$c^\pi(t,t+1) = h_t + \pi(t+1) - \pi(t)$$

avec $\pi(t+1)$ (resp. $\pi(t)$) la valeur du plus court chemin de s à $t+1$ (resp. t) dans le graphe résiduel G_f du flot optimal. On rappelle que $C_{flotPHS}$ est le coût du flot optimal et on pose I_t^f la valeur du flot optimal sur l'arc $(t,t+1)$. Le coût réduit $c^\pi(t,t+1)$ représente la variation de z pour une variation de I'_t d'une unité.

Ainsi dans le cas où $c^\pi(t,t+1) > 0$, on a $I_t^f = 0$ (conditions d'optimalité du flot) et on peut mettre à jour la borne supérieure de I'_t en appliquant :

$$C_{flotPHS} + kc^\pi(t,t+1) > \overline{C'} \Rightarrow \overline{I'_t} < k$$

Si au contraire $c^\pi(t,t+1) < 0$ alors $I_t^f = \overline{I'_t}$ et on peut mettre à jour la borne inférieure de I'_t :

$$C_{flotPHS} - kc^\pi(t,t+1) > \overline{C'} \Rightarrow \underline{I'_t} > \overline{I'_t} - k$$

Dans le cas où $0 < I_t^f < \overline{I'_t}$, le coût réduit est nul et aucun filtrage ne se produit. Les variables de production X'_t sont filtrées de la même manière en utilisant les coûts réduits des arcs de production (s,t).

5.6 Récapitulatif

Le tableau 1 recense les différentes bornes concernées, les valeurs propagées ainsi que les paramètres pris en compte.

Bornes	Valeur	$\overline{X_t}$	$\overline{I_t}$	p_t	h_t	s_t
$\underline{C'}$	$C_{flotPHS}$	√	√	√	√	√
	C_{flotPH} + $\underline{Cs'}$	√	√	√	√	
$\underline{Cp'}$	C_{flotP}	√	√	√		
$\underline{Ch'}$	C_{glouH}	√	√		√	
$\underline{Cs'}$	C_{dynS}	√				√

TABLE 1 – Récapitulatif des bornes

Les algorithmes de filtrage décrits ci-dessus sont utilisés dans l'algorithme de propagation (algorithme 5) de la contrainte globale.

Algorithm 5 Propager

```
1: testerRealisabilite() - Section 4
2: transformerProbleme() - Section 4
3: filtrerCs() - Section 5.2
4: filtrerCh() - Section 5.3
5: filtrerCp() - Section 5.4
6: filtrerC() - Section 5.1
7: filtrerCoutsReduits() - Section 5.5
```

6 Expérimentations

6.1 Résolution du problème par programmation par contraintes et PLNE

Une fois les instants de production décidés (*i.e.* une fois que les Y_t sont fixés), le problème devient facile et est résolu en $O(T^2)$ grâce à l'algorithme de flot sur le réseau G_2 défini à la section 5.1. Il

suffit donc de brancher uniquement sur les variables Y_t. Pour le moment, le branchement est chronologique.

L'implémentation est réalisée en JAVA sur Choco 3.0 [14] sur des instances générées aléatoirement. Le problème traité étant résolu facilement par la PLNE, les benchmarks existants sur le lot-sizing visent plutôt les problèmes avec plusieurs types de produits ou plusieurs niveaux (un produit et ses constituant définissant une nomenclature du produit). Les modèles PLNE sont résolus par ILOG CPLEX 12.5. Deux modèles PLNE sont utilisés à titre de référence : le modèle agrégé (AGG) du *lot sizing* (cf. modèle présenté à la section 3.1) et le modèle *facility location* (FL) [10]. De plus nous renforçons ces modèles en utilisant les bornes obtenues après le point fixe sur les contraintes de flot ce qui renforce leur relaxation linéaire (RL). Bien que la relaxation linéaire du modèle FL soit meilleure que celle du modèle AGG, le grand nombre de variables l'empêche de fournir des solutions en un temps raisonnable. Nous comparons donc seulement les modèles PPC et AGG. Les résultats du modèle simple de PPC consistant à poser la décomposition de la contrainte globale ne sont pas présentés car ce modèle n'est absolument pas compétitif.

6.2 Benchmarking et tests

La génération d'instances se fait de la façon suivante : Les coûts de stockage sont constants et égaux à 1. Les coûts de setup et les coûts unitaires de production sont générés à l'aide de 2 paramètres : e et θ. e représente le coût total unitaire de production (*i.e.* le coût unitaire si la capacité de production est saturée). On le fixe à $e = 10$. $\theta \in [0, 1]$ représente la part du coût de setup par rapport au coût unitaire de production. Le coût de production total à t (*i.e.* e fois la capacité de production) sera donc dû pour θ à son coût de setup et pour $1 - \theta$ à son coût unitaire de production. Pour chaque période, θ est pris aléatoirement de façon uniforme dans un intervalle $[\underline{\theta}, \overline{\theta}]$. La demande est générée selon une loi uniforme dans l'intervalle $[d_{\text{avg}} - \delta, d_{\text{avg}} + \delta]$. Les capacités de production et de stockage sont constantes et peuvent satisfaire $2.5d_{\text{avg}}$. Les cinq classes évaluées sont les suivantes :

- C1 : $d_{\text{avg}} = 1000$, $\delta = 100$, $\theta \in [0.8, 1]$, $T = 40$
- C2 : $d_{\text{avg}} = 1000$, $\delta = 500$, $\theta \in [0.4, 0.6]$,$T = 40$
- C3 : $d_{\text{avg}} = 1000$, $\delta = 100$, $\theta \in [0.8, 1]$, $T = 80$
- C4 : $d_{\text{avg}} = 1000$, $\delta = 500$, $\theta \in [0.4, 0.6]$, $T = 80$
- C5 : $d_{\text{avg}} = 1000$, $\delta = 10$, $\theta = 0.5$, $T = 80$

La limite de temps est fixée à 1 minute d'exécution.

Le tableau 2 présente les résultats par classe et algorithme. Il indique l'écart moyen à la valeur optimale en % (AvgG) de la meilleure solution trouvée, le nombre de preuves d'optimalité effectuées (#R), le temps moyen de résolution en secondes (CPU), le nombre moyen de nœuds explorés (Search) et le nombre de solutions optimales trouvées (#Opt).

	PPC					
CL	T	AvgG	#R	CPU	Search	#Opt
C1	40	0.15 %	0	60.00	72449	5
C2	40	0.30 %	0	60.01	69117	4
C3	80	0.83 %	0	60.01	31015	1
C4	80	0.68 %	0	60.02	27911	0
C5	80	0.13 %	0	60.01	36411	1
	AGG					
CL	T	AvgG	#R	CPU	Search	#Opt
C1	40	0.00 %	10	0.45	1955	10
C2	40	0.00 %	10	0.26	1143	10
C3	80	0.00 %	10	2.07	3663	10
C4	80	0.00 %	10	2.30	3919	10
C5	80	0.00 %	10	3.54	6736	10

TABLE 2 – Comparaison des modèles PPC et AGG

Le tableau 3 montre pour chaque classe d'instances considérée l'écart moyen des différentes bornes sur C par rapport à l'optimal (en %). Les deux dernières colonnes montrent les bornes au nœud racine (BNR) du modèle AGG et sa relaxation linéaire (RL).

CL	$C_{flotPHS}$	$C_{flotPH} + C_{dynS}$	$C_{flotP} + C_{glouH} + C_{dynS}$	AGG BNR	AGG RL
C1	10.67%	13.03%	17.07%	3.13%	10.67%
C2	8.83%	12.89%	16.97%	1.43%	8.83%
C3	9.79%	14.32%	18.55%	2.32%	9.79%
C4	9.03%	14.16%	18.45%	1.77%	9.03%
C5	9.37%	8.57%	8.57%	1.43%	9.37%

TABLE 3 – Comparaison des bornes au nœud racine

On observe bien que $C_{flotP} + C_{glouH} + C_{dynS}$ est dominée par $C_{flotPHS}$. On calcule tout de même cette borne car on a besoin de C_{flotP} et de C_{glouH} pour filtrer *Cp* et *Ch*. Par ailleurs, comme on informe AGG avec les capacités obtenues après la consistance de bornes au nœud racine, la relaxation linéaire de AGG est bien identique à $C_{flotPHS}$. Enfin, on remarque qu'avec des coûts s_t et p_t variables (classes C1 à C4), la borne $C_{flotPHS}$ domine la borne $C_{flotPH} + C_{dynS}$. Cela vient du fait que le calcul des bornes inférieures sur les sous-plans ($w_{t_1 t_2}$) est encore trop faible et prend mal en compte la variabilité des coûts de setup. Ce n'est pas le cas sur C5 où ces coûts sont constants.

Le modèle AGG reste plus efficace à ce stade sur ce problème pur (en particulier pour réaliser la preuve

d'optimalité), les coupes extraites au nœud racine lui donnent une borne initiale très forte.

7 Utilisation de la contrainte globale dans des problèmes multi-items

Cette section illustre l'utilisation de la contrainte globale proposée au travers de trois problèmes multi-produits NP-difficiles. Ces problèmes font tous apparaître des sous-problèmes de *lot sizing* mono-produit avec des contraintes liantes provenant soit des capacités, soit des coûts, soit d'un entrepôt intermédiaire.

Dans chacun, on cherche à planifier la production de N types de produits indicés par $n = 1 \dots N$. Les variables introduites dans le *lot sizing* mono-produit sont maintenant indicées par n. Chaque produit de type n voit une demande $d_{n,t}$ pour $t \in 1 \dots T$. La production d'un produit n induit un coût de setup $s_{n,t}$ et un coût unitaire $p_{n,t}$ en période t. Enfin, le stockage d'un produit n d'une période t à la suivante induit un coût de stockage $h_{n,t}$. Les autres variables sont indicées suivant le même principe.

7.1 Lot sizing multi-produits avec capacité de production commune

Ce premier problème [3] consiste à planifier la production de N types de produits sur une même ligne de production ayant la capacité de produire une quantité maximale $\overline{\alpha_t}$ en période t, tous produits confondus. Le modèle s'écrit :

$C_{global} = \sum_{n=1}^{N} C_n$

(1) $\sum_{n=1}^{N} X_{n,t} \leq \overline{\alpha_t} \qquad \forall\, t = 1 \dots T$

(2) $\textsc{LotSizing}(X_n, I_n, Y_n, Cp_n, Ch_n, Cs_n, C_n, data_n), \quad \forall\, n = 1 \dots N$

(3) $X_n, I_n \in \mathbb{N}^T \qquad \forall\, n = 1 \dots N$

(4) $Y_n \in \{0,1\}^T \qquad \forall\, n = 1 \dots N$

(5) $Cp, Ch, Cs, C \in \mathbb{R}_+^N$

7.2 Joint Replenishment Problem

Dans le Joint Replenishment Problem (JRP), des économies d'échelle sont réalisées si l'on commande (ou fabrique) plusieurs produits à la fois [11]. La production d'au moins un produit en période t engendre un coût de setup *majeur* s_t. On appellera coût de setup *mineur* le coût de setup $s_{n,t}$ associé à la production d'un produit de type n.

Soit Y_t la variable binaire de setup valant 1 si au moins un produit n a été fabriqué à t, 0 sinon. Le modèle s'écrit :

$C_{global} = \sum_{n=1}^{N} C_n + \sum_{t=1}^{T} s_t Y_t$

(1) $\textsc{LotSizing}(X_n, I_n, Y_n, Cp_n, Ch_n, Cs_n, C_n, data_n), \quad \forall\, n = 1 \dots N$

(2) $Y_{n,t} = 1 \Rightarrow Y_t = 1 \qquad \forall\, n = 1 \dots N, \forall\, t = 1 \dots T$

(3) $X_n, I_n \in \mathbb{N}^T \qquad \forall\, n = 1 \dots N$

(4) $Cp, Ch, C, Cs \in \mathbb{R}_+^N$

(5) $Y \in \{0,1\}^T$

7.3 One-Warehouse Multiretailer Problem

Le One-Warehouse Multiretailer Problem (OWMR) est une généralisation du JRP [15]. Un entrepôt central (warehouse) noté 0 approvisionne N détaillants (retailers) indicés par $n = 1 \dots N$ (voir figure 4). Chaque détaillant a les mêmes caractéristiques que le problème mono-produit présenté en section 3. Les quantités commandées par les détaillants sont bornées par les stocks disponibles à l'entrepôt central. Les paramètres et variables caractérisant l'entrepôt sont indicés par 0 : $s_{0,t}$, $p_{0,t}$, $h_{0,t}$, etc. La quantité commandée $X_{0,t}$ par l'entrepôt n'est pas bornée.

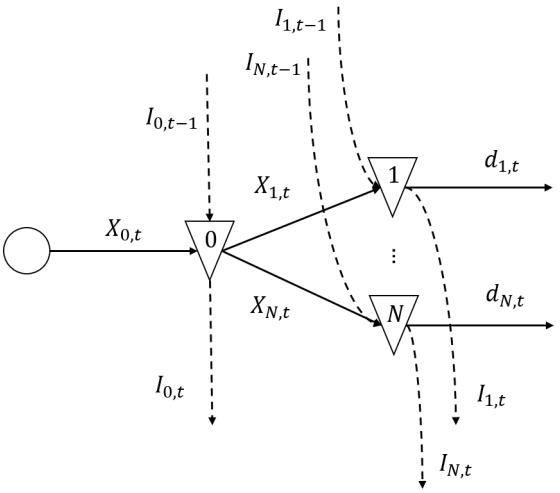


FIGURE 4 – Représentation schématique du problème OWMR à une période t.

En s'appuyant sur la contrainte de LotSizing le modèle s'écrit :

$C_{global} = \sum_{n=1}^{N} C_n + \sum_{t=1}^{T} (p_{0,t}X_{0,t} + s_{0,t}Y_{0,t} + h_{0,t}I_{0,t})$

(2) $X_{0,t} > 0 \Rightarrow Y_{0,t} = 1 \qquad \forall\, t = 1 \dots T$

(3) $I_{0,t-1} + X_{0,t} = \sum_{n=1}^{N} X_{n,t} + I_{0,t} \qquad \forall\, t = 1 \dots T$

(4) $\textsc{LotSizing}(X_n, I_n, Y_n, Cp_n, Ch_n, Cs_n, C_n, data_n) \qquad \forall\, n = 1 \dots N$

(5) $X_n, I_n \in \mathbb{N}^T, Y_n \in \{0,1\}^T \quad \forall\, n = 1 \dots N$

(6) $C, Cp, Ch, Cs \in \mathbb{R}_+^N$

La contrainte (3) est la contrainte de conservation du flot à l'entrepôt.

8 Conclusion et perspectives

Dans un premier temps, nous proposons une contrainte globale pour la résolution d'un problème de *lot sizing* mono-produit très général. Nous étudions la complexité de la contrainte et proposons un cadre (réalisabilité et simplification du problème) pour réaliser le filtrage. Plusieurs bornes inférieures sont données dont une borne équivalente à la relaxation linéaire d'un modèle classique PLNE et une borne originale focalisée sur les coûts de setup. Nos résultats expérimentaux préliminaires valident cette première étape. La suite de l'élaboration de cette contrainte portera sur la généralisation des algorithmes de calcul de bornes et le filtrage des variables de décision qui est encore incomplet. Par ailleurs, nous n'avons pas encore exploré les relaxations des capacités de production et de stockage.

Dans un deuxième temps, nous montrons l'intérêt de cette contrainte pour modéliser de nombreux problèmes multi-produits pour lesquels la PLNE est mise en échec. Nous pensons que la programmation par contraintes peut être plus adaptée pour intégrer les contraintes couplantes. Nous comptons également étudier cette approche sur des réseaux logistiques plus complexes grâce à des reformulations en stocks échelon [19].

Références

[1] Alok Aggarwal and James K. Park. Improved algorithms for economic lot size problems. *Operations Research*, 41(3) :549–571, 1993.

[2] Alper Atamtürk and Simge Küçükyavuz. An $O(n^2)$ algorithm for lot sizing with inventory bounds and fixed costs. *Operations Research Letters*, 36(3) :297–299, 2008.

[3] Imre Barany, Tony J. Van Roy, and Laurence A. Wolsey. Strong formulations for multi-item capacitated lot sizing. *Management Science*, 30(10) :1255–1261, 1984.

[4] Gabriel R. Bitran and Horacio H. Yanasse. Computational complexity of the capacitated lot size problem. *Management Science*, 28(10) :1174–1186, 1982.

[5] Awi Federgruen and Michal Tzur. A simple forward algorithm to solve general dynamic lot sizing models with n periods in $O(n \log n)$ or $O(n)$ time. *Management Science*, 37(8) :909–925, 1991.

[6] Michael Florian and Morton Klein. Deterministic production planning with concave costs and capacity constraints. *Management Science*, 18(1) :12–20, 1971.

[7] Filippo Focacci, Andrea Lodi, and Michela Milano. Cost-based domain filtering. In *Principles and Practice of Constraint Programming–CP99*, pages 189–203. Springer, 1999.

[8] Vinasétan Ratheil Houndji, Pierre Schaus, Laurence Wolsey, and Yves Deville. The stockingcost constraint. In *Principles and Practice of Constraint Programming*, pages 382–397. Springer, 2014.

[9] Jon Kleinberg and Éva Tardos. *Algorithm design*. Pearson Education India, 2006.

[10] Jakob Krarup and Ole Bilde. *Plant Location, Set Covering and Economic Lot Size : An O(mn)-Algorithm for Structured Problems*. Springer, 1977.

[11] Retsef Levi, Robin Roundy, David Shmoys, and Maxim Sviridenko. A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54(4) :763–776, 2008.

[12] Stephen F. Love. Bounded production and inventory models with piecewise concave costs. *Management Science*, 20(3) :313–318, 1973.

[13] Yves Pochet and Laurence A. Wolsey. *Production planning by mixed integer programming*. Springer Science & Business Media, 2006.

[14] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco3 Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2014.

[15] Gautier Stauffer, Guillaume Massonnet, Christophe Rapine, and Jean-Philippe Gayon. A simple and fast 2-approximation algorithm for the one-warehouse multi-retailers problem. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 67–79. SIAM, 2011.

[16] C. P. M. Van Hoesel and Albert Peter Marie Wagelmans. An $O(t^3)$ algorithm for the economic lot-sizing problem with constant capacities. *Management Science*, 42(1) :142–150, 1996.

[17] Albert Wagelmans, Stan Van Hoesel, and Antoon Kolen. Economic lot sizing : an $O(n \log n)$ algorithm that runs in linear time in the wagner-whitin case. *Operations Research*, 40 :S145–S156, 1992.

[18] Harvey M. Wagner and Thomson M. Whitin. Dynamic version of the economic lot size model. *Management science*, 5(1) :89–96, 1958.

[19] Paul Herbert Zipkin. Foundations of inventory management. 2000.

La contrainte globale StockingCost pour les problèmes de planification de production

Vinasetan Ratheil Houndji[1] **Pierre Schaus**[1] **Laurence Wolsey** [1] **Yves Deville** [1]

[1] Université catholique de Louvain, Louvain-la-Neuve, Belgique

{vinasetan.houndji, pierre.schaus, laurence.wolsey, yves.deville}@uclouvain.be

Résumé

Dans beaucoup de problèmes de planification de production, il y a la minimisation des coûts de stockage des articles produits. Le papier [2] introduit la contrainte globale `StockingCost`$([X_1, \ldots, X_n], [d_1, \ldots, d_n], H, c)$ qui est satisfaite quand chacune des demandes X_i est produite avant sa date de livraison d_i, la capacité c de la machine est respectée et H est une borne supérieure sur le coût total de stockage. Ce papier est un résumé de [2] dans lequel est proposé un algorithme linéaire pour faire de la cohérence aux bornes sur la contrainte `StockingCost`. L'efficacité du propagateur `StockingCost` a été démontrée sur un problème de dimensionnement de lots.

Abstract

Many production planning problems call for the minimization of stocking/storage costs. The paper [2] introduces a new global constraint `StockingCost`$([X_1, \ldots, X_n], [d_1, \ldots, d_n], H, c)$ that holds when each item X_i is produced on or before its due date d_i, the capacity c of the machine is respected, and H is an upper bound on the stocking cost. This paper is a summary of [2] that proposes linear time algorithm to achieve bound consistency on the `StockingCost` constraint. On a version of the Discrete Lot Sizing Problem, we demonstrate experimentally the pruning and time efficiency of our algorithm compared to other state-of-the-art approaches.

1 Introduction

Les problèmes de planification de production, comme les problèmes de dimensionnement de lots, consistent à trouver un plan de production d'un ou de plusieurs articles qui satisfait les demandes en respectant la capacité de production des machines et qui minimise les coûts de production.

Pour gérer ce genre de problèmes de dimensionnement de lots, nous proposons la contrainte globale `StockingCost` et un algorithme linéaire pour faire de la cohérence aux bornes.

2 La contrainte StockingCost

La contrainte `StockingCost` a la forme suivante : `StockingCost`$([X_1, \ldots, X_n], [d_1, \ldots, d_n], H, c)$ où :

- la variable X_i est la date de production de la demande i,
- l'entier d_i est la date de livraison pour la demande i,
- l'entier c est la capacité maximum de production,
- si une demande est produite avant sa date de livraison, elle doit être stockée. La variable H est une borne supérieure sur le coût total de stockage.

La contrainte est satisfaite quand toutes les demandes sont produites avant leurs dates de livraison ($X_i \leq d_i$), la capacité totale de production est respectée (on ne peut pas produire plus de c articles à une même période) et H est une borne supérieure sur le coût total de stockage ($\sum_i (d_i - X_i) \leq H$). Example : `StockingCost`$([X_1 \in [1..2], X_2 \in [1..2]], [d_1 = 2, d_2 = 2], H \in [0..2], c = 1)$.

Une première décomposition de la contrainte est :

$X_i \leq d_i, \forall i$ (1)

$\sum_i (X_i = t) \leq c, \forall t$ (2)

$\sum_i (d_i - X_i) \leq H$ (3)

Les inégalités (2) peuvent être remplacées par une contrainte globale `global cardinality constraint (gcc)` (`allDifferent`, dans le cas $c = 1$) [4] ou encore les inégalités (2) et (3) par un `cost-gcc` [5] (`minimum assignment` pour $c = 1$ [1]).

3 Principe de l'algorithme de filtrage

Pour faire de la cohérence aux bornes $BC(H^{min})$ sur la variable de coût H (c'est à dire calculer $H^{opt} = \min \sum_i (d_i - X_i)$), l'idée est de produire les demandes le plus tard possible. Dans l'algorithme décrit dans [2], les demandes sont ordonnées dans l'ordre décroissant de X_i^{max}. Si le problème est faisable (c'est à dire que la contrainte *gcc* est satisfaite) et puisque le coût de stockage est uniforme, il suffit alors de prendre les demandes une à une et de les produire virtuellement à partir de $\max X_i^{max}$ en tenant compte à chaque fois du coût de stockage induit et de la capacité de production.

Si $H^{opt}_{X_i \leftarrow v}$ est le nouveau coût optimal quand X_i prend la valeur v, v doit être supprimée du domaine de X_i si $H^{opt}_{X_i \leftarrow v} > H^{max}$. L'intuition derrière l'agorithme pour faire de la cohérence aux bornes $BC(X_i^{min})$ est de calculer dans un premier temps pour chacune des variables X_i, la valeur ${v_i}^{opt}$ à partir de laquelle H^{min} augmente. Une borne inférieure (mais pas toujours exacte) peut être alors : $v_i^{opt} - (H^{max} - H^{min})$. Dans [2], une procédure a été décrite pour avoir la borne inférieure exacte de X_i en $O(n)$.

4 Résultats expérimentaux sur un problème de dimensionnement de lots

4.1 Description du problème

Le problème de dimensionnement de lots considéré ici, décrit dans [3], est un problème avec plusieurs articles à produire sur une machine de capacité $c = 1$. Il y a des coûts de stockage des articles et des coûts de transition (coût encouru lors du passage de la production d'un article à un autre). Toutes les demandes doivent être satisfaites avant leurs dates de livraison.

4.2 Un modèle en Programmation par Contraintes

Nous identifions de manière unique chaque demande. L'objectif est d'associer à chacune des demandes, une période qui respecte sa date de livraison. Notons $date(p) \in [1..nbPeriods]$, $\forall p \in [1..nbDemands]$ la période dans laquelle la demande p est satisfaite. Si $dueDate(p)$ est la date de livraison de la demande p, on a : $date(p) \leq dueDate(p)$. On peut casser certaines symétries concernant des demandes d'un même article en imposant : $date(p_1) < date(p_2), \forall (p_1, p_2)$ *tel que* $dueDate(p_1) < dueDate(p_2) \wedge item(p_1) = item(p_2)$. En notant $objStorage$ la borne supérieure sur le coût total de stockage, la partie de stockage peut être modélisée par la contrainte : `StockingCost`$(date, dueDate, objStorage, 1)$.

La seconde partie du problème concernant les coûts de transition peut être vue comme un modèle classique du problème du voyageur de commerce où les villes représentent les demandes et la distance entre deux villes est le coût de transition des demandes correspondantes. Si $successor(p)$, $\forall p \in [1..nbDemands]$ est la demande produite sur la machine juste après avoir produit la demande p, on a : $\forall p \in [1..nbDemands]$: $date(p) < date(successor(p))$. Une contrainte `minimum assignement` [1] est ajoutée sur les variables *successor* et les coûts de transition.

4.3 Résultats

Les expériences ont été effectuées avec le solveur OscaR [6]. La comparaison a été faite avec les trois autres décompositions décrites dans la section 2. Sur 15 instances générées aléatoirement, la contrainte `StockingCost` offre plus de filtrage et est plus rapide que les autres décompositions.

5 Conclusion

Pour modéliser certains problèmes de planification de production en Programmation par Contraintes, la contrainte globale `StockingCost` a été proposée ainsi qu'un propagateur efficace.

Références

[1] Filippo Focacci, Andrea Lodi, Michela Milano, and Daniele Vigo. Solving tsp through the integration of or and cp techniques. *Electronic notes in discrete mathematics*, 1 :13–25, 1999.

[2] Vinasetan Ratheil Houndji, Pierre Schaus, Laurence Wolsey, and Yves Deville. The stocking-cost constraint. In *Principles and Practice of Constraint Programming–CP 2014*, pages 382–397. Springer, 2014.

[3] Yves Pochet and Laurence Wolsey. *Production Planning by Mixed Integer Programming*. Springer, 2005.

[4] Claude-Guy Quimper, Peter Van Beek, Alejandro López-Ortiz, Alexander Golynski, and Sayyed Bashir Sadjad. An efficient bounds consistency algorithm for the global cardinality constraint. In *Principles and Practice of Constraint Programming–CP 2003*, pages 600–614. Springer, 2003.

[5] Jean-Charles Régin. Cost-based arc consistency for global cardinality constraints. *Constraints*, 7(3-4) :387–405, 2002.

[6] OscaR Team. Oscar : Scala in or. https ://bitbucket.org/oscarlib/oscar, 2012.

Sur une classe polynomiale relationnelle pour les CSP binaires

Wafa Jguirim[1] **Wady Naanaa**[1] **Martin Cooper**[2] *

[1]Faculté des Sciences, Université de Monastir, Monastir 5000 Tunisie
[2]IRIT, Université de Toulouse, 31062 Toulouse.
jguirimwafa06@gmail.com wady.naanaa@fsm.rnu.tn cooper@irit.fr

Résumé

Trouver une solution à un problème de satisfaction de contraintes est en général un problème NP-difficile. Ainsi, identifier une nouvelle classe polynomiale de CSP, en d'autres termes, une classe de problèmes de satisfaction de contraintes pour lesquelles il existe une méthode de reconnaissance et de résolution de complexité polynomiale, constitue un axe de recherche fondamental dans l'étude des CSPs. Dans cet article, nous présentons une nouvelle classe polynomiale relationnelle pour les CSPs binaires. Tout d'abord nous présentons un nouvel opérateur ternaire que nous appelons mjx. Nous décrivons ensuite les relations fermées par cet opérateur en proposant un algorithme d'identification de ces relations. Afin de réduire la complexité spatiale et temporelle des traitements, nous définissons une nouvelle méthode de stockage de ces relations qui permet de réduire la complexité de chemin consistance, ce dernier niveau de consistance étant suffisant pour résoudre les instances de la classe proposée.

Abstract

Finding a solution to a constraint satisfaction problem is known to be an NP-hard task. Considerable effort has been spent on identifying tractable classes of CSP. In other words, classes of constraint satisfaction problems for which there are polynomial-time recognition and resolution algorithms. In this article, we present a relational tractable class of binary CSP. Our key contribution is a new ternary operation that we named mjx. We first describe mjx-closed relations by proposing an algorithm which identifies such relations. To reduce space and temporal complexity, we define a new storing technique for these relations which reduces the complexity of establishing path consistency, the consistency level that allows solving all instances of the proposed class.

*Martin Cooper est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet TUPLES (ANR-2010-BLAN-0210) ainsi que par l'EPSRC grant EP/L021226/1.

1 Introduction

Dans la vie quotidienne, de nombreux problèmes peuvent être définis en termes de contraintes de temps, d'espace, ou plus généralement de ressources. Nous pouvons citer à titre d'exemples : les problèmes de planification et ordonnancement, ou encore les problèmes d'affectation de ressources. Ces différents problèmes sont désignés par le terme générique CSP *Constraint Satisfaction Problem*, et ont la particularité commune d'être fortement combinatoires : il faut explorer un grand nombre de combinaisons afin de trouver une solution.

Trouver une solution à un problème de satisfaction de contraintes est, en général, un problème NP-difficile. Cependant, un grand nombre de problèmes qui se posent dans la pratique ont des propriétés particulières qui leurs permettent d'être résolus efficacement. Bien que le CSP soit NP-complet, il existe des classes d'instances qui peuvent être à la fois reconnues et résolues en temps polynomial. Dans la littérature, il existe trois types de telles classes, dites polynomiales : les classes structurelles, les classes relationnelles et les classes hybrides. Les classes structurelles sont basées sur la structure du réseau de contraintes. Ces classes exploitent les propriétés topologiques qui existent fréquemment dans les problèmes réels. Par exemple l'acyclicité du réseau [10], ou les réseaux dont la largeur arborescente est bornée par une constante [11]. Les classes relationnelles sont basées sur la sémantique des contraintes, on parle alors de langage de contraintes. Pour les caractériser on utilise l'algèbre des opérations de fermeture (polymorphismes). Parmi les classes relationnelles polynomiales, on peut citer, par exemple les CSP max-fermés [13] et les CSP médiane fermés [12, 8]. Plus récemment, des classes dites hybrides ont

été proposées. Ces classes renferment les deux notions relationnelle et structurelle. On peut citer notamment la classe des instances vérifiant la Broken-Triangle Property [7] et les CSP à rang borné [14].

Dans cette perspective, nous proposons une nouvelle classe polynomiale relationnelle définie sur les relations binaires. Notre contribution consiste à définir un nouvel opérateur que nous appelons mjx. Cet opérateur est un choix très naturel parmi les opérateurs de type majorité. Des avancés récentes [3, 2] ont permis de s'approcher d'une preuve de la conjecture de Feder et Vardi que toute classe relationnelle définie par un ensemble fini de relations est soit polynomiale soit NP-complète [9]. Cependant, d'un point de vue pratique, il reste du travail à faire pour identifier les classes polynomiales qui sont utiles dans de vraies applications. Nous espérons que cet article inspirera d'autres travaux sur la recherche de classes polynomiales vraiment utiles en pratique.

Le reste du papier est organisé comme suit : la Section 2 consiste en une revue bibliographique parcourant les notions, les définitions et notations nécessaires pour la suite. Nous donnerons aussi quelques exemples de relations fermées par mjx. Nous présenterons une caractérisation alternative de telles relations qui nous permettra de donner, dans la Section 3, un algorithme optimal d'identification des relations fermées par cet opérateur. Les instances de CSP fermées par mjx étant résolues en temps polynomial par l'établissement de la 3-consistance forte (voir Section 4), cet article inclut dans la Section 5 un nouvel algorithme de chemin consistance pour les problèmes fermés par mjx. L'algorithme proposé optimise la consistance de chemin moyennant un nouveau calcul des opérations d'intersection et de composition de relations binaires. Contrairement aux anciens travaux [4], les nouveaux algorithmes de calcul de l'intersection et de la composition possèdent une complexité linéaire $O(d)$, où d est la taille du plus grand domaine. Finalement, nous donnons une conclusion sur ce travail dans la Section 6.

2 Préliminaires

Nous rappelons d'abord quelques notions de base sur les CSP et les classes polynomiales.

Définition 1 *Un problème de satisfaction de contraintes (CSP) est défini par un triplet (X, D, C), où*

- *$X = \{x_1, ..., x_n\}$ est un ensemble de n variables.*
- *D est un ensemble fini et totalement ordonné de d valeurs, $D_{x_i} \subseteq D$ (que nous noterons aussi plus simplement D_i) étant l'ensemble des valeurs possibles pour la variable x_i.*
- *$C = \{c_1, ..., c_m\}$ est un ensemble de m contraintes chacune impliquant plusieurs variables.*

Chaque contrainte c_i est un couple $(S(c_i), R(c_i))$, où $S(c_i) \subseteq X$ est la portée de c_i (l'ensemble de variables sur lesquelles elle porte) et $R(c_i) \subseteq D_{j_1} \times \ldots \times D_{j_r}$ (si $S(c_i) = \{x_{j_1}, \ldots, x_{j_r}\}$) est sa relation de compatibilité. L'arité de c_i est le nombre de variables appartenant à $S(c_i)$. En général, on distingue les contraintes binaires, dont l'arité vaut 2, des contraintes d'arité quelconque. Les CSP binaires (dont toutes les contraintes sont binaires) sont généralement considérés de façon différente des CSP dont les contraintes sont d'arité quelconque.

Nous rappelons maintenant la notion de langage de contraintes.

Définition 2 *Un langage de contrainte Γ est un ensemble arbitraire de relations, définies sur un domaine fixe $D(\Gamma)$. On dénote par CSP(Γ) l'ensemble des instances $(X, D(\Gamma), C)$ telles que, pour tout $(S, R) \in C$, on a $R \in \Gamma$.*

Un langage de contraintes fini Γ est dit polynomial s'il existe un algorithme polynomial qui permet de résoudre toutes les instances de CSP(Γ). Un langage de contraintes infini Γ est dit polynomial si chaque sous-ensemble fini de ce langage est polynomial.

Définition 3 *Etant donnée un opérateur $\phi : D^k \to D$, une relation binaire R sur D est fermée par ϕ si, pour tout $(a_1, a'_1), \ldots, (a_k, a'_k) \in D^2$, on a $(a_1, a'_1), \ldots, (a_k, a'_k) \in R \Rightarrow (\phi(a_1, \ldots, a_k), \phi(a'_1, \ldots, a'_k)) \in R$. Un langage Γ est fermé par ϕ si toute relation de Γ est fermée par ϕ.*

Dans ce qui suit nous introduisons notre nouvel opérateur appelé mjx, un opérateur ternaire qui appartient à la classe des opérateurs de type majorité. Cette classe renferme tout les opérateurs ternaires appartenant à la famille des opérateurs de quasi unanimité [12]. Rappelons que $\phi : D^k \to D$ est un opérateur de quasi-unanimité si, pour tout $a, b \in D$, on a

$$\begin{aligned}\phi(b, a, \ldots, a) = \phi(a, b, a, \ldots, a) = \ldots \\ = \phi(a, a, \ldots, b) = a\end{aligned}$$

Définition 4 *L'opérateur* mjx *est défini comme suit :*

$$\text{mjx}(a, b, c) \stackrel{\text{def}}{=} \begin{cases} a & \text{si} \quad a = b \ \vee \ a = c \\ b & \text{si} \quad a \neq b \ \wedge \ b = c \\ \max(a, b, c) & \text{sinon} \end{cases}$$

Ainsi une relation R est fermée par mjx *si et seulement si $(a, a'), (b, b'), (c, c') \in R \Rightarrow (\text{mjx}(a, b, c), \text{mjx}(a', b', c')) \in R$*

Par ailleurs, nous rappelons le théorème suivant. Ce dernier stipule que l'ensemble des CSPs définis sur un langage fermé par un opérateur de majorité forment une classe polynomiale.

Théorème 1 *[12, 3] Soit* Γ *un ensemble de relations sur un domaine fini D. Si* Γ *est fermé par un opérateur de majorité, alors la résolution de CSP* (Γ) *est en temps polynomial.*

Il a été prouvé que la fermeture par un opérateur de majorité est une condition suffisante pour montrer que la résolution d'une instance s'effectue en temps polynomial. En effet, d'une part si une relation R est fermée par un opérateur de majorité ϕ, alors n'importe quelle contrainte (S, R) peut être décomposée en des contraintes binaires et d'autre part toute instance ne comportant que des contraintes fermée par ϕ est résolue par la 3-consistance forte [12, 3].

Corollaire 1 *Soit* $I \in CSP(\Gamma)$. *Si* Γ *est fermé par* mjx *alors la résolution de I s'effectue en temps polynomial.*

Preuve Pour démontrer que mjx est un opérateur de majorité, il suffit d'observer que mjx vérifie la propriété suivante de la famille d'opérateurs de majorité :

$$\forall a, b \quad \text{mjx}(a, a, b) = \text{mjx}(a, b, a) = \text{mjx}(b, a, a) = a$$

L'opérateur mjx est un choix évident parmi tous les opérateurs de majorité puisqu'il retourne le maximum de ses arguments lorsque ces derniers sont tous différents. Une question importante est de savoir s'il existe des relations fermées par mjx qui pourraient arriver en pratique. Nous donnons quelques exemples de telles relations. La vérification que ces relations sont bien fermées par mjx suit presque directement de la caractérisation alternative des relations fermées par mjx que nous donnons dans la Section 3.

Exemple 1 *Toute relation unaire* $x \in A \subset D$ *est fermée par* mjx. *Toute relation binaire sur domaines booléens est fermée par* mjx. *Donc la classe des instances CSP dont toutes les relations sont fermées par* mjx *peut être vue comme une généralisation de 2SAT. Les CSP médiane fermés ("connected row convex") [8] est une autre généralisation de 2SAT qui est incomparable avec les CSP* mjx *fermés.*

Exemple 2 *Soit* f *une fonction monotone décroissante, c'est-à-dire* $u < v \Rightarrow f(u) \geq f(v)$. *Toute relation de la forme* $x \geq f(y)$ *est fermée par* mjx. *Il s'ensuit que les relations binaires suivantes sont toutes fermées par* mjx, *où* x, y *sont les variables et* a, b *des constantes :*

$$x + y \geq a$$
$$(x \geq a) \vee (y \geq b)$$

Les relations suivantes sont aussi fermées par mjx *:*

$$x = y + c$$
$$(x = y + c) \vee (x \geq f(y))$$

Les relations suivantes sont aussi toutes fermées par mjx.

$$(x = a) \vee (y \geq b)$$
$$(x = a) \vee (y = b)$$
$$(x = a) \vee (x \geq f(y))$$
$$(x = a) \vee (y = b) \vee (x \geq f(y))$$
$$((x = a) \wedge (y = b)) \vee (x \geq f(y))$$

Les valeurs (a, b) *pourraient représenter des valeurs par défaut. Par exemple, si* x *est le temps du début d'une action dans un problème de planification,* $x = a$ *pourraient représenter le fait que l'on n'exécute pas l'action en question.*

3 Identification de relations mjx fermées

La manière la plus intuitive de savoir si une relation est fermée par mjx consiste à tester toutes les combinaisons possibles d'éléments qui appartiennent à la relation. Il s'agit de vérifier la fermeture de chaque image de trois paires d'éléments par mjx. Cette méthode simple engendre un nombre de tests de l'ordre de d^6. La section suivante décrit une méthode plus efficace d'identification de relations fermées par mjx.

3.1 Caractérisation des relations fermée par mjx

Nous passons maintenant à des conditions nécessaires et suffisantes de fermeture par mjx. Soit R une relation binaire. Par abus de notation, nous utilisons aussi R pour sa représentation matricielle. La matrice R^* est obtenue à partir de R en supprimant les lignes et les colonnes nulles. Les lignes et colonnes nulles sont naturellement éliminées lorsque l'on établit la consistance d'arc.

Proposition 3.1 *Soit* R *une relation binaire telle que* $R = R^*$ *(c'est-a-dire sans ligne ni colonne nulle). Si* R *est fermée par* mjx *alors*

$$(a' < b') \wedge (a, a'), (a, b') \in R \Rightarrow \forall c' > b', (a, c') \in R \quad (1)$$

et

$$(a < b) \wedge (a, a'), (b, a') \in R \Rightarrow \forall c > b, (c, a') \in R \quad (2)$$

Preuve Puisque $R = R^*$, pour chaque c' il existe c tel que $(c, c') \in R$. Par application directe de la Définition 4, nous obtenons

$$(a, a'), (a, b'), (c, c') \in R \;\Rightarrow\; (a, c') \in R$$

pour tout c' tel que $c' > b' > a'$. De façon analogue, nous obtenons

$$(a, a'), (b, a'), (c, c') \in R \;\Rightarrow\; (c, a') \in R$$

pour tout c tel que $c > b > a$.

Les conditions (1) et (2) garantissent qu'il n'existe pas de zéro précédé par deux 1 au sein d'une même ligne ou d'une même colonne lorsque que la relation est mjx-fermée.

Proposition 3.2 *Si la relation binaire R est fermée par* mjx *alors*

$$\begin{aligned} &((a, a'), (b, b'), (c, c') \in R \wedge \\ &(a > b, c) \;\wedge\; (b' > a', c') \wedge \\ &(c \neq b) \wedge (c' \neq a')) \quad\Rightarrow\quad (a, b') \in R \end{aligned} \tag{3}$$

Preuve Si $(a > b, c)$, $(b' > a', c')$, $c \neq b$ et $c' \neq a'$ alors $(\text{mjx}(a, b, c), \text{mjx}(a', b', c')) = (a, b')$.

Proposition 3.3 *Soit R une relation binaire telle que $R = R^*$. R est fermée par* mjx *si et seulement si nous avons* (1), (2) *et* (3).

Preuve ($\Rightarrow$) Ceci découle des Propositions 3.1 et 3.2. ($\Leftarrow$) Soit une relation binaire R. Si R vérifie (1), (2) et (3) alors quelque soit $(a, b') \notin R$, nous montrerons qu'il n'existe pas $(a, a'), (b, b'), (c, c') \in R$ qui génèrent (a, b') par mjx. Supposons que $\text{mjx}(a, b, c) = a$ et $\text{mjx}(a', b', c') = b'$ et $(a, b') \notin R$. Puisque $(a, a'), (b, b') \in R$ et $(a, b') \notin R$, nous avons $a \neq b$ et $a' \neq b'$. Puisque $\text{mjx}(a, b, c) = a \neq b$, nous avons $c \neq b$ et donc deux possibilités : soit $a > b, c$ soit $a = c$. De façon similaire, puisque $\text{mjx}(a', b', c') = b' \neq a'$ nous avons $c' \neq a'$ et les deux possibilités : soit $b' > a', c'$ soit $b' = c'$. Dans le cas $(a > b, c) \wedge (b' > a', c')$, (3) implique que $(a, b') \in R$ ce qui contredit notre hypothèse. Dans le cas $(a > b, c) \wedge b' = c'$, (2) implique également que $(a, b') \in R$. De façon symétrique, dans le cas $a = c \wedge (b > a', c')$, (1) implique encore que $(a, b') \in R$. Enfin, le cas $a = c \wedge b' = c'$ est impossible car $(c, c') \in R$ et $(a, b') \notin R$. Nous concluons ainsi que R est fermée par mjx.

Proposition 3.4 *On peut stocker une relation binaire R fermée par* mjx *en espace $O(d)$.*

Preuve Ceci est une conséquence de la Proposition 3.1 : au lieu de stocker R sous la forme d'une matrice booléenne, il suffit de stocker les positions des deux premiers 1 dans chaque ligne de cette matrice.

0	0	1	0
0	0	0	1
1	0	0	1
0	1	1	1

2
3
0
1

∞
∞
3
2

FIGURE 1 – Matrice associée à la relation mjx-fermée donnnée dans l'Exemple 3 et les deux vecteurs permettant son stockage.

Exemple 3 *Soit la relation binaire définie sur $\{0, 1, \ldots, d-1\}$ par $(|x - y| = a \;\vee\; x + y > b)$, où a et b sont des constantes telles que $b \leq 2a$. Cette relation est* mjx*-fermée. La matrice associée à une telle relation pour $d = 4$, $a = 2$ et $b = 4$, ainsi que les deux vecteurs qui permettent son stockage selon la Proposition 3.4, sont montrés dans la Figure 1.*

Il est facile, mais fastidieux, de vérifier que les exemples de relations données dans les Exemples 2 et 3 vérifient les conditions (1), (2) et (3) et donc sont bien fermées par mjx. Dans le cas où le domaine de la première variable est booléenne, les conditions (2) et (3) sont toujours vraies de façon triviale (car il n'existe pas trois valeurs a, b, c distinctes dans le domaine de cette variable) et la seule condition à remplir pour être fermé par mjx est la condition (1), ce qui nous permet de donner l'exemple suivant.

Exemple 4 *La relation $(x = a) \Rightarrow (y = b \wedge y \geq c)$, où x est une variable booléenne, y une variable quelconque et a, b, c des constantes, est fermée par* mjx.

3.2 Vérification de la fermeture par mjx

La vérification de fermeture par mjx consiste à vérifier les trois conditions nécessaires et suffisantes de la Proposition 3.3. La méthode la plus simple consiste à vérifier que tous les triplets d'éléments (a, a'), (b, b'), (c, c') qui appartiennent à la relation ne violent pas ces conditions. Malheureusement cette méthode engendre un nombre élevé de tests. Nous montrerons dans cette section qu'il est possible de vérifier les conditions (1), (2), (3) en temps $O(d^2)$.

Dans cette section nous supposons que la relation R à tester se présente sous la forme d'une matrice booléenne : $R(a, b) = 1$ si et seulement si (a, b) appartient à la relation. Il est raisonnable de supposer que convertir une relation donnée sous une autre forme en matrice booléenne peut se faire en temps $O(d^2)$. D'abord, nous pouvons remarquer que la vérification des conditions (1) et (2) nécessite $O(d^2)$ instructions : il suffit de parcourir chaque ligne et chaque colonne une seule fois pour vérifier que le deuxième 1 (s'il existe) de cette ligne ou colonne est suivi par une suite de 1.

Par la suite, dans cette section, nous supposons que $R = R^*$ (c'est-à-dire que les lignes et colonnes nulles de R ont été supprimées) et que R satisfait les conditions (1) et (2). Une conséquence de cette dernière supposition est que chaque zéro $R(i, j) = 0$ est précédé par au plus un 1 dans la ligne i et au plus un 1 dans la colonne j.

Pour vérifier la condition (3), pour chaque (a, b') tel que $R(a, b') = 0$, il faut vérifier qu'il n'existe pas $b, c < a$ et $a', c' < b'$ tels que

$$R(a, a') = R(b, b') = R(c, c') = 1 \\ \wedge (c \neq b) \wedge (c' \neq a') \quad (4)$$

Pour ce faire nous utilisons les structures de données suivantes :

- $NL(i, j) = \sum_{k<j} R(i, k)$ = le nombre de 1 dans la ligne i de R avant la colonne j.
- $NC(i, j) = \sum_{k<i} R(k, j)$ = le nombre de 1 dans la colonne j de R avant la ligne i.
- $N(i, j) = \sum_{k<j} NC(i, k)$ = le nombre total de 1 dans la sous matrice de R composée des lignes avant la ligne i et des colonnes avant la colonne j
- $lig1(j) = \min\{i \mid R(i, j) = 1\}$ = la ligne du premier 1 dans la colonne j de R
- $col1(i) = \min\{j \mid R(i, j) = 1\}$ = la colonne du premier 1 dans la ligne i de R

On peut établir ces structures de données en temps $O(d^2)$ par application directe de leurs définitions, ci-dessus.

Si $R(a, b') = 0$, en s'appuyant sur notre supposition que R satisfait les conditions (1) et (2), nous savons qu'il y a au plus un seul 1 dans la ligne a de R avant la colonne b' et un seul 1 dans la colonne b' avant la ligne a. Il y a donc une seule valeur $a' = col1(a)$ et une seule valeur $b = lig1(b')$ telles que $R(a, a') = R(b, b') = 1$ qui pourraient aussi éventuellement satisfaire $b < a \wedge a' < b'$. Dans le cas où nous avons $b < a \wedge a' < b'$, pour compléter la vérification de la condition (4), il faut vérifier que le nombre de paires (c, c') telles que $R(c, c') = 1 \wedge (c \neq b) \wedge (c' \neq a') \wedge c < a \wedge c' < b'$ soit 0. Le nombre de telles paires (c, c') est donné par la formule suivante

$$N(a, b') - NL(b, b') - NC(a, a') + R(b, a').$$

(le nombre de 1 dans la sous matrice de R composée des lignes avant la ligne a et des colonnes avant la colonne b' moins le nombre de ces 1 qui se trouvent dans la ligne b ou la colonne a'). Nous pouvons conclure que nos structures de données permettent une vérification de la condition (4) en temps $O(1)$ pour chaque paire de valeurs (a, b'), évitant ainsi une recherche complète sur les valeurs de a', b, c et c'. Nous avons démontré la proposition suivante.

Proposition 3.5 *Vérifier qu'une relation R est fermée par* mjx *peut se faire en temps $O(d^2)$.*

On peut même dire que la complexité de $O(d^2)$ est optimale puisque, dans le pire des cas, il faut d^2 opérations pour lire une relation binaire quelconque.

4 Résolution d'instances mjx-fermées

On appelle consistance locale le fait de vérifier que certains sous ensembles de variables ne violent pas les contraintes qui lui sont liées. Cela permet de filtrer alors certaines valeurs ou tuples impossibles avec comme conséquence un coût réduit pour la résolution de l'instance. Il existe plusieurs consistances locales, offrant chacune un équilibre différent entre efficacité du filtrage et rapidité de calcul.

On sait qu'une instance dont toutes les relations sont fermées par un opérateur de majorité, tel que mjx, est résolue soit par la 3-consistance forte [12] soit par la consistance d'arc singleton [5]. Il s'avère que dans le cas des relations fermées par mjx, la consistance de chemin directionnelle et la consistance d'arc suffisent (voir Proposition 4.1). Bien qu'intéressant au niveau théorique, ce résultat ne permet pas forcément de trouver un algorithme plus rapide. Par la suite, nous nous intéresserons plus particulièrement à la 3-consistance forte comme algorithme de résolution de CSPs mjx-fermés.

Définition 5 *Un CSP (X, D, C) est chemin-consistant (directionnel) si pour tout triplet (x_i, x_j, x_k) de variables (tels que $i, j < k$ dans le cas directionnel), et pour toute paire de valeurs (v_i,v_j) $\in D_i \times D_j$, il existe une valeur v_k de D_k telle que l'instanciation partielle ($x_i = v_i$, $x_j = v_j$, $x_k = v_k$) satisfait toutes les contraintes binaires de C portant exclusivement sur x_i, x_j et x_k.*

Proposition 4.1 *Soit I une instance CSP binaire dans laquelle toutes les contraintes sont* mjx*-fermées. Il suffit d'établir simultanément la consistance de chemin directionnelle et la consistance d'arc pour déterminer si I est satisfiable et pour pouvoir trouver une solution, s'il en existe, en temps linéaire.*

Preuve Supposons que I est arc consistante et chemin consistante directionnelle. Soit $(a_1, \ldots, a_{k-1})$ une solution partielle, c'est-à-dire une affectation aux variables $(x_1, \ldots, x_{k-1})$ qui satisfait toutes les contraintes portant sur ces variables. Par consistance d'arc, nous savons qu'une telle solution partielle existe pour $k = 3$. Nous montrerons qu'il est toujours possible (pour $k = 3, \ldots, n$) d'étendre $(a_1, \ldots, a_{k-1})$ à une solution partielle $(a_1, \ldots, a_k)$. Par récurrence, il en suivra que I

est satisfiable et que l'on peut trouver une solution en temps linéaire.

Notons R_{ij} la relation de la contrainte portant sur la paire de variables (x_i, x_j). Soit $R_{ik}(a_i)$ l'ensemble des valeurs $b \in D_k$ telle que $(a_i, b) \in R_{ij}$. Grâce à la consistance d'arc, nous savons que la représentation matricielle de R_{ij} ne comporte ni ligne ni colonne nulle. On peut donc déduire de la Proposition 3.1 que $R_{ik}(a_i)$ est de la forme

$$\{p_{ik}\} \ \cup \ \{v \mid s_{ik} \leq v \leq m_k\}$$

où $m_k = \max(D_k)$, $p_{ik} < s_{ik}$, $p_{ik} \in D_k$ et $s_{ik} \in D_k \cup \{\infty\}$ (p_{ik} et s_{ik} étant respectivement les positions du premier et du deuxième 1 dans la ligne a_i de la matrice R_{ij}). Par consistance de chemin, $\forall i, j$ tels que $1 \leq i, j < k$ et $i \neq j$, on a

$$R_{ik}(a_i) \cap R_{jk}(a_j) \ \neq \ \emptyset \qquad (5)$$

Il y a deux cas :

1. $\forall i = 1, \ldots, k-1$, $m_k \in R_{ik}(a_i)$, ou
2. $\exists i$ tel que $s_{ik} = \infty$ et donc $R_{ik}(a_i) = \{p_{ik}\}$.

Dans le premier cas, on peut choisir $a_k = m_k$. Dans le deuxième cas, on peut choisir $a_k = p_{ik}$ car $p_{ik} \in R_{jk}(a_j)$ (pour tout $j = 1, \ldots, k-1$) par (5). Dans les deux cas, $(a_1, \ldots, a_k)$ est une solution partielle.

5 Optimisation de la 3-consistance forte

L'algorithme ci-dessous décrit une procédure simple pour établir la 3-consistance forte, où AC établit la consistance d'arc.

Algorithm 1 Procédure 3C_1 (X, D, C)

```
répéter
    pour tout x_i, x_j, x_k ∈ X faire
        si (i ≠ j) ∧ (i ≠ k) ∧ (j ≠ k) alors
            R_ij ← R_ij ∩ Π_ij (R_ik ⋈ R_kj ⋈ D_k)
        fin si
    fin pour
    AC(X, D, C)
jusqu'à (pas de changement)
```

La procédure 3C_1 montre l'intérêt de pouvoir effectuer rapidement les opérations d'intersection $R \cap S$ et de composition $\Pi_{x,y}$ $(R \bowtie S)$. Dans les deux sous sections suivantes nous montrons que ces deux opérations peuvent s'effectuer en temps $O(d)$ si les relations R et S sont fermées par mjx, au lieu de $O(d^2)$ et $O(d^3)$ respectivement pour des relations quelconques.

5.1 L'intersection de deux relations mjx fermées

L'opération d'intersection s'applique sur des paires de relations comme suit :

$$R \cap S = \{(u, v) \in D^2 \mid (u, v) \in R \ \wedge (u, v) \in S\}$$

Nous utiliserons par la suite l'opération $min\_2(E)$ qui retourne la deuxième plus petite valeur de l'ensemble E. Nous supposons que chaque relation R est stockée, comme évoqué dans la proposition 3.4, au moyen des deux variables suivantes :

- $col1_R(i) = \min\{j \mid R(i,j) = 1\}$ = la colonne du premier 1 dans la ligne i de R.
- $col2_R(i) = \min\_2\{j \mid R(i,j) = 1\}$ = la colonne du deuxième 1 dans la ligne i de R.

Dans le cas où le premier 1 ou le deuxième 1 n'existe pas, la variable associée, $col1_R(i)$ ou $col2_R(i)$, prend par défaut une valeur supérieure à d que nous noterons ∞. En s'appuyant sur le fait que le deuxième 1 dans une ligne est suivi par une suite de 1, calculer l'intersection $T = R \cap S$ peut se faire à partir des règles suivantes : $\forall i$,

$$col1_T(i) \ = \ \begin{cases} col1_R(i) & \text{si } col1_R(i) = col1_S(i) \\ col1_R(i) & \text{si } col1_R(i) \geq col2_S(i) \\ col1_S(i) & \text{si } col1_S(i) \geq col2_R(i) \\ \max(col2_R(i), col2_S(i)) & \text{sinon} \end{cases}$$

et

$$col2_T(i) \ = \ \begin{cases} \max(col2_R(i), col2_S(i)) & \\ \qquad \text{si } col1_R(i) = col1_S(i) & \\ col2_S(i) & \text{si } col1_S(i) \geq col2_R(i) \\ col2_R(i) & \text{si } col1_R(i) \geq col2_S(i) \\ \max(col2_R(i), col2_S(i)) + 1 & \text{sinon} \end{cases}$$

Pour simplifier la présentation, ici nous supposons que $D_i = \{1, \ldots, d\}$ avec $d + 1 = \infty$. Pour chaque valeur de i, ces calculs s'effectuent en temps constant. Donc le calcul de $R \cap S$ s'effectue en temps $O(d)$. Puisqu'il faut retourner les $2d$ valeurs $col1_T(i)$, $col2_T(i)$ qui représentent la relation T, on peut en déduire que cette complexité est optimale.

5.2 Composition de deux relations fermées par mjx

Nous focalisons dans cette section sur l'opération de la composition qui opère sur une paire de relations R et S comme suit :

$$R \circ S = \{(u, v) \in D^2 \mid \exists w \in D, (u, w) \in R \wedge (w, v) \in S\}$$

On sait que les polymorphismes sont conservés sous les opérations de projection et de jointure [6]. Donc, si R et S sont fermées par mjx, alors $T = R \circ S$ le sera aussi. Il s'ensuit que l'on peut représenter T au moyen

des valeurs $col1_T(i)$, $col2_T(i)$ (pour chaque ligne i de la représentation matricielle de T).

Par souci de simplicité de présentation, nous supposons que les matrices R et S sont toutes les deux de taille $d \times d$.

Pour faciliter le calcul nous utiliserons les structures de données suivantes :

- $m1_R(i) = \min\{j \mid \exists k \geq i, R(k,j) = 1\}$ = la première colonne j telle qu'il existe un 1 dans au moins une des lignes $k \geq i$
- $m2_R(i) = \min\_2\{j \mid \exists k \geq i, R(k,j) = 1\}$ = la deuxième colonne j telle qu'il existe un 1 dans au moins une des lignes $k \geq i$.

Dans l'algorithme de composition, donné ci-dessous, la première boucle calcule de façon incrémentale les valeurs de $m1_S(i)$ et $m2_S(i)$ pour chaque ligne i, en commençant avec la dernière ligne.

Pour calculer la ligne i de la matrice $T = R \circ S$, nous distinguons trois cas, selon que la ligne i de R a la forme (1) $(0, \ldots, 0)$, (2) $(0, \ldots, 0, 1, 0, \ldots, 0)$ ou (3) $(0, \ldots, 0, 1, 0, \ldots, 0, 1, \ldots, 1)$. Dans chaque cas, on peut déduire la forme de la ligne i de T directement de la définition de l'opération de composition.

1. Si la ligne i de R est nulle, alors la ligne i de T sera nulle aussi.
2. Si la ligne i de R ne comporte qu'un seul 1 (dans la colonne j), alors la ligne i de T sera identique à la ligne j de S.
3. Si la ligne i de R comporte au moins deux 1 (dont les deux premiers dans les colonnes j et $k > j$), alors $col1_T(i) = \min\{col1_S(j), m1_S(k)\}$ et $col2_T(i) = \min\_2(col1_S(j), col2_S(j), m1_S(k), m2_S(k))$.

L'algorithme 2 reçoit, en entrée, deux relations R et S qui sont supposées être fermées par mjx et donc stockées sous la forme des positions des deux premiers 1 dans chaque ligne ($col1_R$ et $col2_R$ pour R et $col1_S$ et $col2_S$ pour S). L'Algorithme calcule, alors, la composition selon les trois règles données ci-dessus (deuxième boucle **pour**). Au préalable, il remplit les structures de données $m1_S$ et $m2_S$ décrites ci-dessus (première boucle **pour**). Le résultat de l'algorithme, recevant en entrée R et S, sont deux vecteurs qui mémorisent, respectivement, la position du premier et du deuxième 1 dans chaque ligne de la matrice associée la relation $R \circ S$.

La complexité de notre algorithme de composition de relations fermées par mjx est $O(d)$. Puisqu'il faut retourner les $2d$ valeurs $col1_T(i)$, $col2_T(i)$, on peut en déduire que cette complexité est optimale. Cependant, l'utilisation des algorithmes optimaux d'intersection et de composition des relations fermées par mjx dans un algorithme simple de chemin consistance, tel que 3C_1, n'est pas optimale dans le pire cas. Dans la sous section suivante nous montrerons comment construire un algorithme de chemin consistance pour les instances dont toutes les relations sont fermées par mjx, de complexité temporelle maximale $O(n^3 d \log d + n^2 d^2)$ et de complexité spatiale $O(n^2 d)$.

Algorithm 2 Fonction *Composition*$(R, S) : T = R \circ S$

```
- - R et S sont données sous la forme des vecteurs col1_R, col2_R et col1_S,col2_S
- - Le résultat, T = R ∘ S, est calculée dans col1_T et col2_T
 1: m_1 ← ∞
 2: m_2 ← ∞
 3: pour i de d à 1 faire
 4:     m1_S(i) ← min(col1_S(i),m_1)
 5:     m2_S(i) ← min_2(col1_S(i),col2_S(i),m_1,m_2)
 6:     m_1 ← m1_S(i)
 7:     m_2 ← m2_S(i)
 8: fin pour
 9: pour i de 1 à d faire
10:     j ← col1_R(i)
11:     k ← col2_R(i)
12:     si (k = ∞) alors
13:         si (j = ∞) alors - - cas 1 : ligne i de R nulle
14:             col1_T(i) ← ∞
15:             col2_T(i) ← ∞
16:         sinon  - - cas 2 : ligne i de R contient un 1
17:             col1_T(i) ← col1_S(j)
18:             col2_T(i) ← col2_S(j)
19:         fin si
20:     sinon  - - cas 3 : ligne i de R comporte deux 1
21:         E ← {col1_S(j), col2_S(j), m1_S(k), m2_S(k)}
22:         col1_T(i) ← min(E)
23:         col2_T(i) ← min_2(E)
24:     fin si
25: fin pour
26: retourner (col1_T, col2_T)
```

5.3 Algorithme de 3-consistance forte

L'algorithme de composition de la Section 5.2 montre l'utilité des structures de données $m1_S$ et $m2_S$. Nous proposons de stocker ces structures de données pour chaque relation S de l'instance, au lieu de les recalculer chaque fois que l'on souhaite recalculer la composition de deux relations R et S. Pendant l'établissement de la 3-consistance forte, les valeurs $col1_S(k)$ et $col2_S(k)$ ne peuvent qu'augmenter (suite à la suppression d'un 1 dans la ligne i de la relation S). Pour mettre à jour les valeurs de $m1_S(i)$ et $m2_S(i)$ lorsque $col1_S(k)$ ou $col2_S(k)$ augmente, il suffit de reprendre la première boucle de l'algorithme de composition à partir de la ligne $i = k$ et de continuer l'exécution de cette boucle tant que $m1_S(i)$ ou $m2_S(i)$ a changé par rapport à son ancienne valeur. Ainsi le nombre total d'opérations sera de l'ordre du nombre de mises à jour

des variables $col1_S$, $col2_S$ plus le nombre des mises à jour des variables $m1_S$, $m2_S$, ce qui représente un total de $O(d^2)$ opérations par relation, car chacune de ces quatre variables est monotone et ne peut prendre que $O(d)$ valeurs.

Lorsque $col1_R(i)$ ou $col2_R(i)$ augmente, pour mettre à jour la relation $T = R \circ S$, il faut recalculer $col1_T(i)$ ou $col2_T(i)$ en exécutant l'itération i de la deuxième boucle de l'algorithme de composition.

Il faut aussi recalculer $col1_T(i)$ ou $col2_T(i)$, en exécutant l'itération i de cette même boucle, dans les deux cas suivants :

1. lorsque $col1_S(j)$ ou $col2_S(j)$ augmente où $j = col1_R(i)$,
2. lorsque $m1_S(k)$ ou $m2_S(k)$ augmente où $k = col2_R(i)$.

Dans ces deux cas, nous utilisons des structures de données dédiées pour avoir un accès rapide aux valeurs de i telles que $j = col1_R(i)$ ou $k = col2_R(i)$:

$$\begin{aligned} \forall j \quad inv\_col1_R(j) &= \{i \mid col1_R(i) = j\} \\ \forall k \quad inv\_col2_R(k) &= \{i \mid col2_R(i) = k\} \end{aligned}$$

Les structures de données inv_col1_R et inv_col2_R nécessitent en tout seulement $O(d)$ espace mémoire par relation. Nous pouvons remarquer que le maintien de ces structures de données inverses peut se faire en temps $O(\log d)$ par opération d'ajout ou de retrait (ces opérations étant nécessaires lorsque $col1_R(i)$ ou $col2_R(i)$ augmente pour une valeur de i) en utilisant, par exemple, un tas [1]. Le nombre total de ses mises à jour sera $O(d^2)$ par relation, ce qui donne une complexité totale de $O(n^2d^2 \log d)$ pour le maintien des structures de données inv_col1_R et inv_col2_R.

Le nombre maximum de fois qu'une relation portant sur deux variables x_i, x_j peut être modifiée est $O(d^2)$. Après chaque modification, il faut rétablir la 3-consistance sur le triplet x_i, x_j, x_k pour chaque troisième variable x_k, ce qui entraîne une complexité totale de $O(n^3d^2 + n^2d^2 \log d)$ pour ces propagations (y compris le maintien des structures de données inv_col1_R et inv_col2_R). L'algorithme détaillé est donné dans la Section 5.4.

Pendant l'établissement de la 3-consistance forte, il faut aussi établir et maintenir la consistance d'arc. Le nombre total d'opérations pour maintenir la consistance d'arc est $O(n^2d^2)$ si on garde en mémoire, par exemple, le minimum support pour chaque affectation (x_i, v) dans le domaine D_j de chaque autre variable x_j [4]. Cette structure de données ne nécessite que $O(n^2d)$ espace mémoire. Il s'ensuit que la consistance d'arc ne fait pas augmenter la complexité (temporelle ou spatiale) maximale. Donc, dans le cas où toutes les relations sont fermées par mjx, établir la 3-consistance forte (et donc résoudre l'instance) peut se faire en complexité temporelle $O(n^2d^2(n + \log d))$ et en complexité spatiale $O(n^2d)$.

5.4 Algorithme complet de 3-consistance forte

Dans cette sous section, nous donnons en détail l'algorithme décrit dans la Section 5.3. L'Algorithme 3 de 3-consistance forte appelle les deux sous programmes MAJ_élim_vals et Propager_PC, décrits plus loin, jusqu'à la convergence. La liste L_1 est une liste de paires (i, u) telles que la valeur u est à éliminer du domaine de la variable x_i. Lorsque l'on effectue cette élimination, il faut mettre à jour toute relation R_{ij} ainsi que toute relation R_{ji}. On rappelle qu'une relation R est stockée sous la forme des structures de données $col1_R$, $col2_R$, $m1_R$ et $m2_R$, que nous noterons $col1_{ij}$, $col2_{ij}$, $m1_{ij}$ et $m2_{ij}$ lorsque $R = R_{ij}$. Les domaines D_i sont stockés explicitement mais aussi implicitement dans les relations R_{ij} puisque les lignes et les colonnes nulles correspondent respectivement aux valeurs $u \notin D_i$ et aux valeurs $v \notin D_j$.

Les sous programmes MAJ_col12 et MAJ_m12 mettent à jour respectivement les structures de données $col1$, $col2$ et $m1$, $m2$. MAJ_col12(i,j,l,c,L_1,L_2) effectue une mise à jour éventuelle de $col1_{ij}(l)$ et $col2_{ij}(l)$ lorsque (l, c) est éliminé de la relation R_{ij}. Il est utilisé pour s'assurer que (l, c) soit éliminé de R_{ij} lorsque (c, l) est éliminé de R_{ji}. L'algorithme MAJ_m12 reprend la première boucle de l'algorithme de composition de la Section 5.2.

Algorithm 3 Procédure 3-consistance_forte(X, D, C)

```
1: L1 ← {(i, a) | (x_i ∈ X) ∧ (a ∉ D_i)}
2: L2 ← {(i, j, a) | (x_i, x_j ∈ X) ∧ (a ∈ D_i)}
3: répéter
4:     MAJ_élim_vals(L1, L2)
5:     Propager_PC(L1, L2)
6: jusqu'à (L1 = ∅) ∧ (L2 = ∅)
```

L'algorithme Propager_PC(L_1, L_2) propage les éléments de la liste L_2 : chaque élément (i, j, l) appartenant à la liste L_2 y a été ajouté suite à un changement qui est survenu concernant la ligne l de la relation R_{ij} (c'est-a-dire dans les valeurs de $col1_{ij}$, $col2_{ij}$, $m1_{ij}$ ou $m2_{ij}$). Pour chaque troisième variable $k \neq i, j$, il faut rétablir la consistance de chemin sur les deux triplets de variables : (x_i, x_j, x_k) et (x_k, x_i, x_j). Ceci se fait de façon optimisée selon la méthode indiquée dans la Section 5.3. Les mises à jour des relations R_{ik} et R_{kj}, suite au rétablissement de la consistance de chemin, peuvent entraîner des ajouts à la liste L_1 indiquant les valeurs qui peuvent être éliminées par la consistance d'arc. La liste L_1 est traitée par le sous programme MAJ_élim_vals.

Algorithm 4 Procédure MAJ_élim_vals(L_1,L_2)

répéter
  $(i, u) \leftarrow \text{pop}(L_1)$
  si ($u \in D_i$) **alors**
    $D_i \leftarrow D_i \setminus \{u\}$
    pour $j \in X \setminus \{i\}$ **faire**
      $col1_{ij}(u) \leftarrow \infty$
      $col2_{ij}(u) \leftarrow \infty$
      MAJ_m12(i,j,u,L_2)
      pour $w \in D_j$ **faire**
        MAJ_col12(j,i,w,u,L_1,L_2)
      fin pour
    fin pour
  fin si
jusqu'à ($L_1 = \emptyset$)

Algorithm 5 Procédure MAJ_col12(i,j,l,c,L_1,L_2)

si ($col1_{ij} = c$) **alors**
  $col1_{ij}(l) \leftarrow col2_{ij}(l)$
  Ajouter (i, j, l) à L_2
  si ($col1_{ij}(l) = \infty$) **alors**
    Ajouter (i, l) à L_1
  sinon
    $col2_{ij}(l) \leftarrow \min\{b \mid (b > c) \wedge (b \in D_j \cup \{\infty\})\}$
  fin si
  MAJ_m12(i,j,l,L_2)
sinon
  si ($col2_{ij}(l) = c$) **alors**
    $col2_{ij}(l) \leftarrow \min\{b \mid (b > c) \wedge (b \in D_j \cup \{\infty\})\}$
    MAJ_m12(i,j,l,L_2)
  fin si
fin si

Algorithm 6 Procédure MAJ_m12(i,j,l,L_2)

$k \leftarrow l$
si ($k = d$) **alors**
  $m_1 \leftarrow \infty$
  $m_2 \leftarrow \infty$
sinon
  $m_1 \leftarrow m1_{ij}(k + 1)$
  $m_2 \leftarrow m2_{ij}(k + 1)$
fin si
répéter
  $change \leftarrow$ FAUX
  $m1_{ij}(k) \leftarrow min(col1_{ij}(k),m_1)$
  $m2_{ij}(k) \leftarrow min\_2(col1_{ij}(k),col2_{ij}(k),m_1,m_2)$
  si au moins une de $m1_{ij}$,$m2_{ij}$ a changée **alors**
    $change \leftarrow$ VRAI
    Ajouter (i, j, k) à L_2
  fin si
  $m_1 \leftarrow m1_{ij}(k)$
  $m_2 \leftarrow m2_{ij}(k)$
  $k \leftarrow k - 1$
jusqu'à ($k = 0$) $\vee$ ($change$ = FAUX)

Le sous programme MAJ_PC(i,j,l,k,L_1,L_2) effectue des mises à jour de la ligne l de la relation R_{ik} suite au changement dans la relation R_{ij} ou la relation R_{jk}. Il s'inspire directement de l'algorithme d'intersection de la Section 5.1 et de l'algorithme de composition de la Section 5.2. Il appelle MAJ_m12 pour mettre à jour $m1_{ik}$, $m2_{ik}$ et il appelle MAJ_col12 pour mettre à jour la relation transposée R_{ki}.

Algorithm 7 Procédure MAJ_PC(i,k,l,j,L_1,L_2)

- - recalcul de la ligne l de $R_{ij} \bowtie R_{jk}$:
$c_1 \leftarrow col1_{ij}(l)$
$c_2 \leftarrow col2_{ij}(l)$
si ($k = \infty$) **alors**
  si ($j = \infty$) **alors**
    $col1 \leftarrow \infty$
    $col2 \leftarrow \infty$
  sinon
    $col1 \leftarrow col1_{jk}(c_1)$
    $col2 \leftarrow col2_{jk}(c_1)$
  fin si
sinon
  $E \leftarrow \{col1_{jk}(c_1), col2_{jk}(c_1), m1_{jk}(c_2), m2_{jk}(c_2)\}$
  $col1 \leftarrow \min(E)$;
  $col2 \leftarrow \min\_2(E)$
fin si
- - mise à jour de la ligne l de R_{ik} :
si ($col1 \geq col2_{ik}(l)$) **alors**
  $col1_{ik}(l) \leftarrow col1$
  $col2_{ik}(l) \leftarrow col2$
sinon
  si ($col1_{ik}(l) = col1$) **alors**
    $col2_{ik}(l) \leftarrow col2$
  sinon
    $col1_{ik}(l) \leftarrow col2$
    $col2_{ik}(l) \leftarrow \min\{c \mid c > col2 \wedge c \in D_k \cup \{\infty\}\}$
  fin si
fin si
si $col1_{ik}(l)$ devient ∞ **alors** Ajouter (i, l) à L_1
fin si
MAJ_m12(i,k,l,L_2)
pour tout (l, c) éliminé de R_{ik} **faire**
  MAJ_col12(k,i,c,l,L_1,L_2)
fin pour

6 Conclusion

Dans cet article, nous avons étudié une classe relationnelle polynomiale, la classe des instances CSP binaires dans lesquelles toutes les relations sont fermées par l'opérateur mjx. Cet opérateur de type majorité est une fonction ternaire qui retourne le maximum de ces arguments lorsque ceux-ci sont tous distincts. Nous avons montré, via des exemples, que certaines relations utiles sont fermée par mjx. Nous avons aussi donné

Algorithm 8 Procédure Propager_PC(L_1,L_2)

répéter
  $(i, j, l) \leftarrow \text{pop}(L_2)$
  pour tout $x_k \in X \setminus \{x_i, x_j\}$ **faire**
    - - pour la ligne l de R_{ik},
    - - $R_{ik} \leftarrow R_{ik} \cap \Pi_{ik}(R_{ij} \bowtie R_{jk}))$:
    MAJ_PC(i,k,l,j,L_1,L_2)
    pour $(h \in inv\_col_{ij}(l) \cup inv\_col2_{ij}(l))$ **faire**
      - - pour la ligne h de R_{kj},
      - - $R_{kj} \leftarrow R_{kj} \cap \Pi_{kj}(R_{ki} \bowtie R_{ij}))$:
      MAJ_PC(k,j,h,i,L_1,L_2)
    fin pour
  fin pour
jusqu'à ($L_2 = \emptyset$)

une caractérisation alternative de relations fermées par mjx ce qui nous a permis de démontrer que de telles relations peuvent être stockées en espace $O(d)$. Une autre contribution était la réalisation d'un algorithme optimal d'identification de relations fermées par mjx de complexité $O(d^2)$.

L'exploitation de structures de données pour stocker les relations fermées par mjx dépasse le gain mémoire puisqu'elle nous a permis de proposer deux nouveaux algorithmes d'une complexité linéaire $O(d)$ de calcul de l'intersection et de la composition de deux relations fermées par mjx. Par conséquent, nous avons optimisé la complexité temporelle de la 3-consistance forte qui passe de $O(n^3d^3)$ à $O(n^2d^2(n + \log d))$ pour une complexité spatiale de $O(n^2d)$. La 3-consistance forte est une méthode de résolution des CSP binaires dont toutes les relations sont fermées par mjx.

Ces résultats montrent que l'étude d'une classe relationnelle polynomiale définie à partir d'un choix judicieux de polymorphisme peut se révéler intéressante en termes d'une méthode de stockage compact et d'un algorithme de résolution efficace. Il serait intéressant dans l'avenir d'étudier d'autres classes relationnelles définies par des polymorphismes simples dans l'espoir de trouver d'autres résultats similaires, voire même d'essayer d'établir une théorie concernant les classes relationnelles qui peuvent être résolues en temps borné par un polynôme donné. Par exemple, on peut se demander s'il existe un algorithme de résolution de complexité $o(n^3d^3)$ pour CSP(Γ) pour tout langage Γ fermé par un opérateur de majorité. Une autre perspective de recherche théorique serait d'étudier le polymorphisme de type quasi-unanimité d'arité $k > 3$ qui retourne le maximum de ses arguments sauf dans le cas où au moins $k - 1$ de ces arguments sont égaux (le cas $k = 3$ correspondant au polymorphisme mjx).

Il serait intéressant de chercher des applications dans lesquelles toutes les relations sont fermées par mjx.

Références

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[2] Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1) :3, 2014.

[3] Libor Barto, Marcin Kozik, and Ross Willard. In *LICS*, pages 125–134.

[4] Christian Bessière, Jean-Charles Régin, Roland H. C. Yap, and Yuanlin Zhang. An optimal coarse-grained arc consistency algorithm. *Artif. Intell.*, 165(2) :165–185, 2005.

[5] Hubie Chen, Víctor Dalmau, and Berit Grußien. Arc consistency and friends. *J. Log. Comput.*, 23(1) :87–108, 2013.

[6] David A. Cohen and Peter G. Jeavons. The complexity of constraint languages. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, pages 245–280. Elsevier, 2006.

[7] Martin C. Cooper, Peter G. Jeavons, and András Z. Salamon. Generalizing constraint satisfaction on trees : Hybrid tractability and variable elimination. *Artif. Intell.*, 174(9-10) :570–584, 2010.

[8] Yves Deville, Olivier Barette, and Pascal Van Hentenryck. Constraint satisfaction over connected row convex constraints. *Artif. Intell.*, 109(1-2) :243–271, 1999.

[9] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction : A study through Datalog and group theory. *SIAM J. Comput.*, 28(1) :57–104, 1998.

[10] Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32(4) :755–761, October 1985.

[11] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2) :243–282, 2000.

[12] Peter Jeavons, David A. Cohen, and Martin C. Cooper. Constraints, consistency and closure. *Artif. Intell.*, 101(1-2) :251–265, 1998.

[13] Peter Jeavons and Martin Cooper. Tractable constraints on ordered domains. *Artif. Intell.*, 79(2) :327–339, février 1995.

[14] Wady Naanaa. Unifying and extending hybrid tractable classes of csps. *J. Exp. Theor. Artif. Intell.*, 25(4) :407–424, 2013.

De nouvelles approches pour la décomposition de réseaux de contraintes *

Philippe Jégou **Hanan Kanso** **Cyril Terrioux**

Aix-Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296
13397 Marseille Cedex 20 (France)
{philippe.jegou, hanan.kanso, cyril.terrioux}@lsis.org

Résumé

Selon la nature des instances de CSP à traiter, les méthodes exploitant la décomposition (arborescente) offrent une approche souvent efficace pour la résolution, le comptage de solutions et l'optimisation. Aussi, une bonne partie des efforts de la communauté s'est focalisée sur l'élaboration d'algorithmes visant à trouver les meilleures décompositions, i.e. celles qui minimisent la largeur arborescente, soit le paramètre central en termes de complexité théorique. Dans ce cadre, l'heuristique $Min\text{-}Fill$ constitue la méthode de référence. Nous introduisons ici deux nouvelles méthodes heuristiques dont le but est d'améliorer $Min\text{-}Fill$. L'une a pour objet de fonctionner sans *triangulation*, ce qui permet d'améliorer les temps de calculs. L'autre vise à améliorer le comportement de $Min\text{-}Fill$ en exploitant mieux la topologie des graphes. L'évaluation expérimentale que nous présentons montre l'intérêt de ces nouvelles approches.

Abstract

Depending on the nature of CSP instances to consider, the (tree-)decomposition methods offer an approach often efficient for the solving, the counting of solutions or the optimization. So, the community has focused a large part of its efforts on the production of algorithms aiming to find the best decompositions, i.e. ones which minimize the *tree-width*, a central parameter in terms of theoretical complexity. In this frame, the heuristic $Min\text{-}Fill$ constitutes the reference method. We introduce here two new heuristics with the aim in view to improve $Min\text{-}Fill$. The first one aims to proceed without *triangulation*, allowing notably an improvement of the runtime. The second intends to improve the behaviour of $Min\text{-}Fill$ by exploiting the topology of graphs. The experimental evaluation we present shows the interest of these new approaches.

*Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet TUPLES (ANR-2010-BLAN-0210).

1 Introduction

Les Problèmes de Satisfaction de Contraintes (CSP) offrent un cadre puissant pour la formulation de problèmes en Informatique, et en particulier en Intelligence Artificielle. Formellement, un *Problème de Satisfaction de Contraintes* est un triplet (X, D, C), où $X = \{x_1, \ldots, x_n\}$ est un ensemble de n variables, $D = \{D_{x_1}, \ldots, D_{x_n}\}$ est un ensemble de domaines finis de valeurs, et $C = \{c_1, \ldots, c_e\}$ est un ensemble fini de e contraintes. Chaque contrainte c_i est une paire $(S(c_i), R(c_i))$, où $S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ définit la *portée* de c_i, et $R(c_i) \subseteq D_{x_{i_1}} \times \cdots \times D_{x_{i_k}}$ est sa *relation de compatibilité*. L'*arité* de c_i est notée $|S(c_i)|$. Un CSP est dit *binaire* si toutes ses contraintes sont d'arité 2. La structure d'un réseau de contraintes (autre terme pour appeler un CSP) est représentée par un hypergraphe (qui est un graphe dans le cas binaire), appelé *(hyper)graphe de contraintes*, et dont les sommets correspondent aux variables et les arêtes aux portées des contraintes. Pour simplifier les notations, dans la suite, nous noterons l'hypergraphe $(X, \{S(c_1), \ldots S(c_e)\})$ par (X, C). Sans manque de généralité, nous supposerons que les réseaux considérés sont connexes. Une affectation sur un sous-ensemble de X sera dite *cohérente* si elle ne viole aucune contrainte. Vérifier si un CSP possède une *solution* (i.e. une affectation cohérente de toutes les variables) est bien connu pour constituer un problème NP-complet. Aussi, de nombreux travaux ont été développés pour améliorer la résolution en pratique. Bien évidemment, la complexité de ces approches demeure exponentielle, au moins en $O(n.d^n)$ où n est le nombre de variables et d la taille maximum des domaines. Au-delà, étant donnée une instance CSP, d'autres questions peuvent être formulées conduisant à des problèmes plus difficiles

encore comme l'optimisation sous contraintes (NP-difficile) [4] ou le comptage de solutions (#P-complet) [20]. De plus, on peut utiliser certaines techniques développées dans le cadre CSP pour traiter des problèmes de compilation de connaissances qui conduisent également à des problèmes très difficiles en termes de complexité [10].

Afin de contourner la complexité intrinsèque de ces différents problèmes (décision, optimisation, comptage, compilation), d'autres approches sont fondées sur l'exploitation des classes polynomiales structurelles qui s'appuient sur la notion de *décomposition arborescente* de graphes [16]. Leur avantage fondamental est lié à leur complexité théorique, en général de l'ordre de d^{w+1} où w est la *largeur arborescente* du graphe de contraintes [1]. Ainsi, quand la largeur est limitée, ces méthodes permettent de traiter des instances de grande taille. C'est le cas par exemple des problèmes bien connus d'optimisation pour l'allocation de fréquences radio [5]. Notons qu'en pratique, la complexité en temps est plutôt de l'ordre de $d^{w^+ +1}$ où $w^+ \geq w$ est une approximation de la largeur arborescente car le calcul de décompositions optimales (i.e. de largeur w) constitue un problème NP-difficile [1].

Toutefois, la mise en œuvre pratique de ce type de méthodes, bien qu'elle ait très souvent démontré son intérêt, a également permis de constater que la minimisation du paramètre w^+ n'est pas toujours la plus appropriée comme nous l'avions montré lors des JFPC 2014 [14] et de CP 2014 [13]. Cela étant, lorsque l'on s'intéresse à des problèmes plus difficiles comme l'optimisation, le comptage ou encore la compilation, la largeur et donc sa minimisation devient alors cruciale. Malheureusement l'obtention d'une largeur optimale semble inaccessible en pratique, sauf pour de très petits graphes (quelques dizaines de sommets tout au plus) [2]. Aussi, en pratique, les décompositions sont généralement calculées par des approches heuristiques pour lesquelles *Min-Fill* [17] constitue à ce jour le meilleur compromis entre temps de calcul et qualité de la décomposition obtenue. Pour autant, cette méthode recèle certains inconvénients. D'une part, elle procède par triangulation, c'est-à-dire, par ajout d'arêtes. Cela peut rendre cette approche extrêmement coûteuse en temps, voire inopérante pour le cas de graphes de grande taille. D'autre part, *Min-Fill* n'exploite pas explicitement les propriétés topologiques du graphe, ce qui de fait pose un problème justement dans le cas du calcul de décompositions et conduit parfois cette heuristique à l'obtention de décompositions de qualité médiocre. La complexité temporelle de *Min-Fill* est $O(n(n+e'))$ où e' est le nombre d'arêtes du graphe après triangulation (on a donc toujours $e' \geq e$).

Dans cette contribution, afin de contourner ces problèmes, nous introduisons de nouvelles approches heuristiques. La première, appelée *Least-TD*, opère sans triangulation en réalisant un calcul basé sur un parcours du graphe qui se limite au calcul de clusters par le biais d'une exploitation de propriétés liées aux séparateurs (un séparateur est un ensemble de sommets dont la suppression engendre la présence de plusieurs composantes connexes dans le graphe) et aux composantes connexes. Elle conduit à un algorithme dont la complexité temporelle est inférieure à celle de *Min-Fill* avec $O(n(n+e))$, attestée d'ailleurs par une meilleure efficacité pratique. De plus, cette approche permet d'améliorer dans de nombreux cas la qualité de la décomposition.

La seconde approche peut être considérée comme mixte au sens où elle utilise les qualités de *Min-Fill* tout en exploitant la topologie du graphe mais en cherchant aussi à éviter les cas de figure où *Least-TD* pose problème. Si elle fournit également des décompositions qui sont en général de meilleure qualité que celles calculées par *Min-Fill*, l'exploitation conjointe de la triangulation et de la topologie du graphe conduit à une complexité plus élevée. En effet, l'algorithme *Min-Fill-MG* qui la met en œuvre a une complexité de l'ordre de $O(n^2(n+e'))$.

La partie suivante introduit les notions de bases associées à la décomposition arborescente et aux méthodes permettant d'en calculer. La partie 3 introduit l'algorithme *Least-TD* alors que la partie 4 présente l'algorithme *Min-Fill-MG*. Avant de conclure, nous présentons des résultats expérimentaux.

2 La décomposition et ses calculs

Historiquement, le *Tree-Clustering* (noté *TC* [8]) est la méthode de référence pour la résolution de CSP binaires via l'exploitation de la structure de leur graphe de contraintes. Elle est basée sur la notion de *décomposition arborescente de graphes* [16].

Définition 1 *Étant donné un graphe $G = (X, C)$, une* décomposition arborescente *de G est une paire (E, T) où $T = (I, F)$ est un arbre et $E = \{E_i : i \in I\}$ une famille de sous-ensembles (appelés* clusters*) de X, telle que chaque cluster E_i est un nœud de T et vérifie :*

(i) $\cup_{i \in I} E_i = X$,

(ii) pour chaque arête $\{x, y\} \in C$, il existe $i \in I$ avec $\{x, y\} \subseteq E_i$, et

(iii) pour tout $i, j, k \in I$, si k est sur un chemin de i vers j dans T, alors $E_i \cap E_j \subseteq E_k$.

1. Cette notion est exploitée pour les hypergraphes en l'appliquant sur leur *2-section* [3]. La 2-section d'un hypergraphe (X, C) est le graphe (X, C') tel que $C' = \{\{x, y\} | \exists c \in C, \{x, y\} \subseteq c\}$.

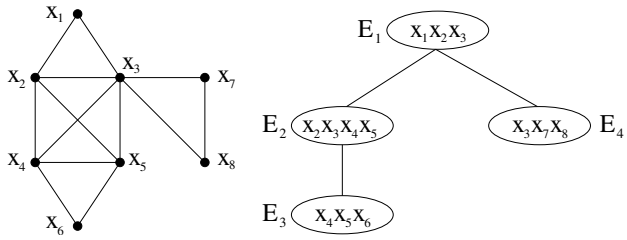


FIGURE 1 – Un graphe de contraintes de 8 variables (a) et une décomposition arborescente optimale (b).

Notons que la condition (iii) peut être remplacée par : si un sommet x est tel que $x \in E_i \cap E_j$, alors, tous les nœuds E_k de T qui apparaissent dans l'unique chemin de E_i à E_j contiennent x. La largeur d'une décomposition arborescente (E, T) est égale à $max_{i \in I}|E_i|-1$. La largeur arborescente *ou* tree-width *w de G est la largeur minimale pour toutes les décompositions arborescentes de G.*

La figure 1(b) présente un arbre dont les nœuds correspondent aux cliques maximales du graphe présenté dans la figure 1(a). Il s'agit de l'une des décompositions arborescentes de ce graphe. Ainsi, nous avons $E_1 = \{x_1, x_2, x_3\}$, $E_2 = \{x_2, x_3, x_4, x_5\}$, $E_3 = \{x_4, x_5, x_6\}$, et $E_4 = \{x_3, x_7, x_8\}$. La taille maximum des clusters dans cette décomposition arborescente optimale vaut 4 et donc, la largeur arborescente de ce graphe vaut 3.

La première version de TC [8] débute par le calcul d'une décomposition arborescente qui utilise l'algorithme MCS (Maximum Cardinality Search [19]). Dans la seconde étape, les clusters sont résolus indépendamment, en considérant chaque cluster comme un sous-problème, et donc, en énumérant toutes ses solutions. Après cela, une solution globale au CSP, s'il en existe une, peut alors être efficacement calculée en exploitant la structure arborescente de la décomposition. Le complexité en temps de cette première version est en $O(n.w^+.log(d).d^{w^+ +1})$ et en $O(n.d^{w^+ +1})$ pour l'espace, où $w^+ + 1$ est la taille du plus grand cluster $(w + 1 \leq w^+ + 1 \leq n)$.

Notons que cette première approche a été améliorée pour arriver à une complexité en espace en $O(n.s.d^s)$ [7, 6, 12] où s est la taille de la plus grande intersection (*séparateur*) entre deux clusters ($s \leq w^+$). Aussi, pour rendre ces méthodes structurelles efficaces, il faut *a priori* minimiser les valeurs de w^+ et s lors du calcul de la décomposition arborescente. Ceci est d'autant plus crucial quand on considère, au-delà des problèmes de décision, les problèmes d'optimisation, de comptage ou encore de compilation.

Malheureusement, calculer une décomposition arborescente optimale (i.e. de largeur w) est NP-difficile [1]. Aussi, de très nombreux travaux ont abordé cette question. Ils exploitent souvent une approche algorithmique fondée sur la notion de graphe *triangulé* (voir [11] pour une introduction à cette classe de graphes), sachant que pour ces graphes, calculer une décomposition arborescente optimale est linéaire. Nous pouvons distinguer différentes classes d'approches. Tout d'abord, il y a les méthodes exactes ou par approximation (avec garantie) mais qui n'ont pas démontré à ce jour leur intérêt pratique pour une raison de temps d'exécution extrêmement important au regard de la faible amélioration de la valeur de w^+ qu'elles permettent d'obtenir. Viennent ensuite les méthodes heuristiques n'offrant aucune garantie en termes d'optimalité (comme celles fondées sur les *triangulations heuristiques*). Ce sont elles qui sont les plus utilisées en pratique. Elles opèrent en temps polynomial (entre $O(n+e)$ et $O(n^3)$), sont très simples à implémenter, et leur avantage semble tout à fait justifié. En effet, ces heuristiques semblent obtenir des triangulations assez proches de l'optimum [15]. C'est le cas notamment de *Min-Fill* [17] qui calcule souvent des décompositions optimales en un temps beaucoup plus court que les approches exactes [18]. Aussi, en pratique, *Min-Fill* constitue l'approche de référence depuis maintenant plusieurs décennies. Nous la présentons dans le détail.

Algorithme 1 : *Min-Fill*

Entrées : Un graphe $G = (X, C)$
Sorties : Un graphe G' triangulation de G et un *peo*
1 Calcul du nombre d'arêtes nécessaires pour la complétion du voisinage de chaque sommet
2 **tant que** $\exists$ *un sommet non numéroté* **faire** `/* choix du prochain sommet dans l'ordre */`
3 Choix d'un sommet v non numéroté nécessitant un minimum d'arêtes pour compléter son voisinage ultérieur
4 Numéroter v
5 Compléter le sous-graphe induit par les voisins de v non numérotés
6 Mettre à jour le nombre d'ajouts d'arêtes nécessaires pour chaque sommet non numéroté

Min-Fill procède en numérotant et donc en ordonnant les sommets de G de 1 à n tout en rajoutant les arêtes nécessaires de sorte que cet ordre soit un *ordre d'élimination parfait* (ou *peo*) pour le graphe résultant G' qui sera triangulé. Dans un tel ordre, les voisins ultérieurs de chaque sommet (c'est-à-dire les voisins apparaissant après ce sommet dans l'ordre) forment une clique. La triangulation va être réalisée sur un graphe G' initialisé à G. À chaque étape, *Min-Fill* choisit parmi les sommets non numérotés, le sommet qui a besoin du minimum d'arêtes à rajouter pour compléter le sous-graphe induit [2] par ses voisins non encore ordonnés. Le processus continue jusqu'à ce que tous les sommets de G soient numérotés.

2. Le sous-graphe $G[Y]$ d'un graphe $G = (X, C)$ induit par $Y \subseteq X$ est le graphe (Y, C_Y) avec $C_Y = \{\{x, y\} \in C | x, y \in Y\}$.

La complexité en temps de *Min-Fill* est $O(n(n+e'))$ où e' le nombre d'arêtes du graphe triangulé G'. Si l'heuristique *Min-Fill* s'avère être la plus coûteuse parmi les heuristiques de l'état de l'art, son temps d'exécution est bien meilleur que celui des méthodes exactes, tout en calculant des décompositions assez proches de l'optimum. Cependant, *Min-Fill* souffre de plusieurs défauts. D'une part, elle procède d'une certaine façon à l'aveugle en se limitant à des décomptes d'arêtes sans se préoccuper des propriétés topologiques du graphe traité, du moins, explicitement. D'autre part, l'ajout d'arêtes occasionné par le principe de triangulation génère un coût temporel important. Aussi, dans les sections suivantes, nous introduisons deux nouvelles heuristiques visant à effacer ces défauts.

3 Décomposition sans triangulation

Nous proposons ici un algorithme appelé *Least-TD*, qui calcule, pour un graphe $G = (X, C)$, une décomposition arborescente en temps polynomial, bien sûr sans aucune garantie quant à son optimalité, mais sans recours à la triangulation et en prenant en compte la topologie du graphe. L'objectif est double : obtenir de meilleurs temps de calcul et éviter certains défauts de *Min-Fill* en exploitant d'éventuelles propriétés topologiques.

D'une certaine façon, *Least-TD* s'inspire de l'algorithme *BC-TD* [14, 13] qui décompose également sans triangulation. La première étape de l'algorithme calcule un premier cluster, noté E_0. X' qui notera par la suite l'ensemble des sommets déjà traités est donc initialisé à E_0. Cette première étape peut se faire facilement, en utilisant une méthode heuristique. Notons $X_1, X_2, \dots X_k$ les composantes connexes du sous-graphe $G[X \backslash E_0]$ induit par la suppression dans G des sommets de E_0. Chacun de ces ensembles X_i est inséré dans une file F. Pour chaque élément X_i supprimé de F, on notera V_i l'ensemble des sommets de X' qui sont adjacents à au moins un sommet de X_i. On peut noter que V_i est un séparateur du graphe G puisque la suppression de V_i dans G rend G non connexe (X_i étant déconnecté du reste de G). Nous considérons alors le sous-graphe de G induit par V_i et X_i, c'est-à-dire $G[V_i \cup X_i]$. L'étape suivante va consister à déterminer un sommet v de V_i qui possède un minimum de voisins dans X_i. Soit $N(v, X_i) = \{x \in X_i : \{v, x\} \in C\}$ cet ensemble. On construit alors un nouveau cluster $E_i = N(v, X_i) \cup V_i$.

La figure 2 présente le calcul de E_1, le second cluster (après E_0), lors du premier passage dans la boucle. Parmi les sommets de V_1, celui qui possède un minimum de voisins dans X_1 est le sommet v puisque l'on a $N(v, X_1) = \{a\}$ et que pour les autres sommets, cet ensemble est $N(w, X_1) = N(x, X_1) = \{b, c\}$. Le cluster E_1 aura ainsi pour sommets $V_1 \cup \{a\}$. À partir de là, on va obtenir deux nouvelles composantes connexes : $X_{1_1} = \{d, e, h\}$ et $X_{1_2} = \{b, c, f, g, i, j, k\}$ qui vont alors être ajoutées à la file F. Quand X_{1_1} en sera retiré, on aura $V_i = \{a\}$, un nouveau cluster sera alors construit avec $V_i \cup \{d, e\} = \{a, d, e\}$ et $\{h\}$ sera ajouté à F. Quand X_{1_2} sera retiré de la file, on considérera l'ensemble $V_i = \{a, w, x\}$ pour lequel le sommet choisi sera a qui ne possède qu'un voisin dans X_{1_2} pour former un nouveau cluster $\{a, f, w, x\}$. Deux nouveaux ensembles seront alors insérés dans F : $\{i\}$ et $\{b, c, g, j, k\}$. On remarque qu'ainsi, l'algorithme exploite explicitement la topologie du graphe par le biais de séparateurs et de composantes connexes.

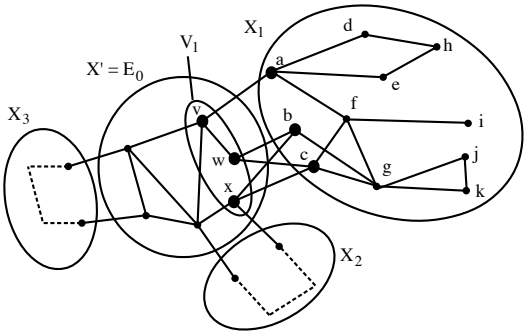


FIGURE 2 – Illustration de *Least-TD*

Algorithme 2 : *Least-TD*

Entrées : Un graphe $G = (X, C)$
Sorties : Un ensemble de clusters $E_0, \dots E_m$ d'une décomposition arborescente de G

1 Choix d'un premier cluster E_0 dans G
2 $X' \leftarrow E_0$
3 Soient $X_1, \dots X_k$ les composantes connexes de $G[X \backslash E_0]$
4 $F \leftarrow \{X_1, \dots X_k\}$
5 **tant que** $F \neq \emptyset$ **faire** `/* calcul d'un nouveau cluster` E_i `*/`
6 Enlever X_i de F
7 Soit $V_i \subseteq X'$ le voisinage de X_i dans G
8 Déterminer un sommet $v \in V_i$ qui possède un minimum de voisins $N(v, X_i)$ dans X_i
9 $E_i \leftarrow N(v, X_i) \cup V_i$
10 $X' \leftarrow X' \cup N(v, X_i)$
11 Soient $X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}$ les composantes connexes de $G[X_i \backslash E_i]$
12 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$

La file F va être exploitée jusqu'à la suppression de l'ensemble des sommets du graphe. Nous établissons maintenant la validité de l'algorithme, puis sa complexité temporelle.

Théorème 1 *Least-TD calcule les clusters d'une décomposition arborescente.*

Preuve : Il suffit de prouver la correction des lignes 5 à 12 de l'algorithme. Nous montrons d'abord que l'al-

gorithme s'arrête. À chaque passage dans la boucle, puisque $N(v, X_i)$ est ajouté à X', au moins un sommet de X_i sera rajouté à l'ensemble X' et ce sommet n'apparaîtra pas plus tard dans un nouvel élément de la file d'attente puisque ces éléments sont définis par les composantes connexes de $G[X_i \backslash E_i]$, un sous-graphe qui contient strictement moins de sommets qu'il n'y en a dans X_i. Ainsi, après un nombre fini d'étapes, l'ensemble $X_i \backslash E_i$ sera un ensemble vide, et donc aucun nouvel ajout à F ne sera possible.

Nous montrons maintenant que l'ensemble des clusters $E_0, E_1, \dots E_m$ induit bien une décomposition arborescente de G. Nous le prouvons par une induction sur l'ensemble des clusters ajoutés en montrant que tous ces clusters vont induire une décomposition arborescente du graphe $G[X']$. Initialement, le premier cluster E_0 induit une décomposition arborescente du graphe $G[E_0] = G[X']$. L'hypothèse d'induction est que l'ensemble des clusters déjà ajoutés $E_0, E_1, \dots E_{i-1}$ induit une décomposition arborescente du graphe $G[E_0 \cup E_1 \cup \dots \cup E_{i-1}]$. Considérons maintenant l'ajout de E_i. Nous montrons que par construction, $E_0, E_1, \dots E_{i-1}$ et E_i induit une décomposition arborescente du graphe $G[X']$ en montrant que les trois conditions (i), (ii) et (iii) de la définition des décompositions arborescentes sont satisfaites.

(i) Chaque nouveau sommet ajouté dans X' appartient à E_i

(ii) Chaque nouvelle arête de $G[X']$ est à l'intérieur du cluster E_i.

(iii) On peut considérer deux cas différents pour un sommet $x \in E_i$, sachant que pour les autres sommets, la propriété est déjà satisfaite par l'hypothèse d'induction. Si $x \in V_i$, la propriété a déjà été vérifiée par hypothèse d'induction. Si $x \in E_i \backslash V_i = N(v, X_i)$, x n'apparaît pas dans un autre cluster que E_i et la propriété est vérifiée.

Finalement, il est facile de voir que l'on obtient bien les clusters d'une décomposition arborescente du graphe $G[X']$, et par extension de G puisqu'en fin de traitement, on a $X' = X$. □

Théorème 2 *La complexité en temps de l'algorithme Least-TD est $O(n(n+e))$.*

Preuve : Les lignes 1-4 sont réalisables en temps linéaire, soit $\mathrm{O}(n+e)$, puisque le coût de calcul des composantes connexes de $G[X \backslash E_0]$ est borné par $\mathrm{O}(n+e)$. Pour le coût de la boucle (ligne 5), notons qu'il y a nécessairement moins de n insertions dans la file F car à chaque passage dans la boucle, on est assuré qu'au moins un nouveau sommet aura été rajouté dans X', et donc supprimé de l'ensemble des sommets n'ayant pas encore été traités. Nous analysons maintenant le coût de chaque traitement associé à l'ajout d'un nouveau cluster, dont nous donnons pour chacun, sa complexité globale.

— Ligne 6 : l'obtention du premier élément X_i de F est bornée par $\mathrm{O}(n)$, soit globalement $\mathrm{O}(n^2)$.
— Ligne 7 : l'obtention du voisinage $V_i \subseteq X'$ de X_i dans G est bornée par $\mathrm{O}(n+e)$, et donc globalement par $\mathrm{O}(n(n+e))$.
— Ligne 8 : cette étape est réalisable en $\mathrm{O}(n)$, soit globalement en $\mathrm{O}(n^2)$. En effet, on peut profiter du traitement de la ligne 7 pour calculer pour tous les sommets de V_i le nombre de sommets voisins que chacun possède dans X_i, cela n'engendrant aucun surcoût au niveau de ce traitement.
— Lignes 9 et 10 : chacune de ces étapes est réalisable en $\mathrm{O}(n)$, soit globalement en $\mathrm{O}(n^2)$.
— Ligne 11 : le coût de la recherche des composantes connexes de $G[X_i \backslash E_i]$ est borné par $\mathrm{O}(n+e)$. Ainsi le coût de cette étape est $\mathrm{O}(n(n+e))$.
— Ligne 12 : l'insertion de X_{i_j} dans F est réalisable en $\mathrm{O}(n)$, soit globalement en $\mathrm{O}(n^2)$ puisqu'il y a moins de n insertions dans F.

Finalement, la complexité temporelle de l'algorithme *Least-TD* est $\mathrm{O}(n(n+e))$. □

D'un point de vue pratique, on peut supposer que le choix du premier cluster E_0 peut être crucial pour la qualité de la décomposition qui est en cours de calcul. Nous pouvons d'ailleurs noter que ce choix peut être réalisé par une heuristique éventuellement plus coûteuse, de sorte à obtenir un premier cluster plus pertinent, mais en se limitant à un coût de l'ordre de $\mathrm{O}(n(n+e))$ afin de ne pas accroître la complexité globale de l'algorithme. De même, le choix du sommet v sélectionné dans la ligne 8, peut être d'une importance considérable sur le plan pratique quand plusieurs sommets sont éligibles. Des heuristiques peuvent bien sûr être utilisées mais cette question ne sera pas abordée dans le cadre de ce travail.

Dans la section suivante, nous proposons un algorithme de triangulation basé à la fois sur *Min-Fill* et sur l'approche de l'algorithme *Least-TD*.

4 Trianguler en exploitant la topologie

4.1 L'effet boule de neige

Outre le fait que *Min-Fill* ne garantit pas une triangulation minimum ni même minimale [15], cette heuristique présente un phénomène bien connu qui est *l'effet boule de neige*. Il est observé au niveau de l'ajout considérable d'arêtes au graphe à trianguler (potentiellement d'ordre quadratique par rapport au nombre

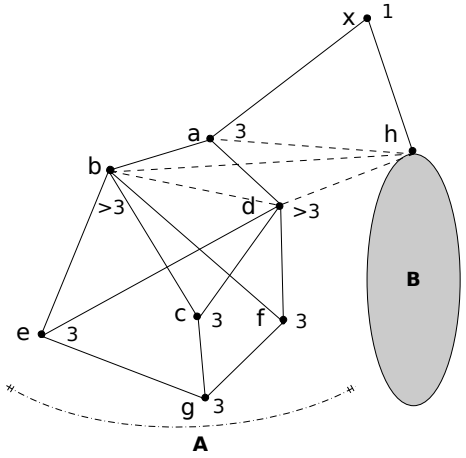


FIGURE 3 – Illustration de *l'effet boule de neige*

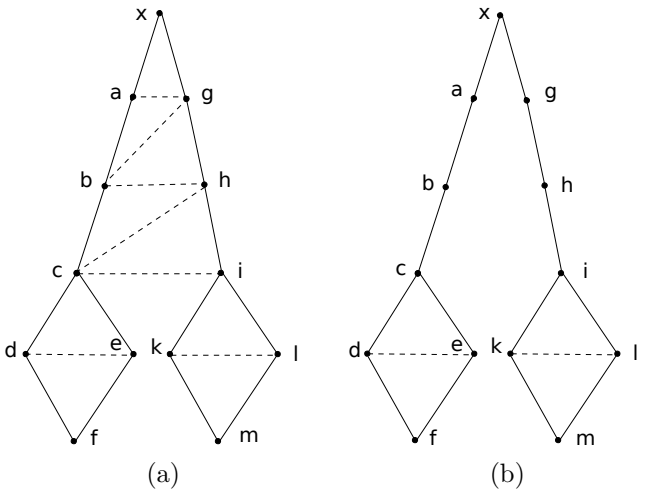


FIGURE 4 – Triangulation ne respectant pas la topologie (a) et une triangulation respectant la topologie (b)

de sommets considérés comme c'est le cas par exemple pour une grille ou *grid graph*). Souvent, cet aspect est dû à un ajout d'arêtes ne respectant pas ce qui serait souhaitable au regard de la topologie du graphe. En effet, la démarche suivie par *Min-Fill* ignore la topologie du graphe et ne tient compte que du nombre d'arêtes nécessaires pour compléter le voisinage d'un sommet. Ainsi, la présence de *l'effet boule de neige* est tout à fait possible avec *Min-Fill*.

On illustre ce phénomène par la figure 3. Les arêtes pleines sont les arêtes du graphe original et les arêtes en pointillés sont les arêtes ajoutées par la triangulation. Ce graphe est constitué de deux parties indépendantes A et B, en considérant le sommet x comme séparateur. Deux parties A et B sont *indépendantes* si aucune arête n'est présente entre un sommet de A et un sommet de B. Ce sont les deux composantes connexes induites par la suppression du sommet x dans le graphe (si l'on suppose qu'aucun sommet de B ne constitue un meilleur choix). La partie B est grisée par souci de simplification. Chaque sommet est annoté par le nombre d'arêtes à ajouter pour compléter son voisinage. Compte tenu de l'heuristique de choix, *Min-Fill* choisit le sommet nécessitant le minimum d'ajout d'arêtes, soit x dans ce cas. En choisissant le sommet x, on ajoute l'arête $\{a, h\}$. Une fois mis à jour, les sommets du graphe auraient toujours besoin du même nombre d'arêtes à rajouter. Désormais, A et B forment une seule composante connexe. Ce phénomène est susceptible de se reproduire si on choisit à l'étape suivante le sommet a, ce qui conduira à rajouter les arêtes $\{b, h\}$ et $\{d, h\}$ en plus de $\{b, d\}$. De plus, au moment de la prise en compte de h, cela pourrait amener à rajouter des arêtes entre les nouveaux voisins de h dans A et les voisins qu'il possède déjà dans B, augmentant ainsi significativement le nombre d'arêtes ajoutées. La rétroaction permettrait donc de renforcer *l'effet de boule de neige*. Une solution à ce problème consiste à traiter indépendamment d'abord les sommets de A ou de B.

La figure 4 montre deux triangulations possibles d'un graphe. Elle montre que la triangulation qui respecte la topologie du graphe ajoute moins d'arêtes que la version qui ne tient pas compte de cette topologie. En effet, *Min-Fill* établit un ordre débutant par x dans la figure 4(a) avec $O = [x; a; g; b; h; f; d; e; c; i; k; l; m]$. Cependant, dans la figure 4(b) *Min-Fill* réussit à ajouter moins d'arêtes en commençant par la partie A avec un ordre $O = [f; d; e; c; b; a; x; g; h; i; k; l; m]$.

Les inconvénients de *l'effet boule de neige* ne se limitent pas à l'ajout excessif d'arêtes qui augmentent le temps de calcul de la triangulation et plus généralement le temps d'obtention de la décomposition arborescente. Il peut y avoir aussi un impact sur la largeur arborescente w de la décomposition calculée. En effet, l'ajout excessif d'arêtes augmente potentiellement la taille des cliques du graphe en cours de triangulation, et par voie de conséquence, la taille des clusters de la décomposition, donc sa largeur.

4.2 Mieux guider Min-Fill

Dans cette partie, nous proposons une version nouvelle de *Min-Fill* appelée *Min-Fill-MG* et qui permet d'exploiter la topologie du graphe à trianguler. Cette approche applique une stratégie assez semblable à celle employée par *Least-TD*. Nous utiliserons d'ailleurs les mêmes notations. En effet, comme *Least-TD*, l'algorithme commence par le choix d'un

Algorithme 3 : *Min-Fill-MG*

Entrées : Un graphe $G = (X, C)$
Sorties : Un ensemble de clusters $E_0, \dots E_m$ d'une décomposition arborescente de G

1 Choix d'un premier cluster E_0 dans G
2 $X' \leftarrow E_0$
3 Soient $X_1, \dots X_k$ les composantes connexes de $G[X \backslash E_0]$
4 $F \leftarrow \{X_1, \dots X_k\}$
5 **tant que** $F \neq \emptyset$ **faire** `/* calcul d'un nouveau cluster` E_i `*/`
6 Enlever X_i de F
7 Soit $V_i \subseteq X'$ le voisinage de X_i dans G
8 Complétion de V_i dans G
9 $G_i \leftarrow Minfill(G[V_i \cup X_i])$
10 $CM \leftarrow$ Cliques maximales de G_i
11 $E_i \leftarrow$ Clique de CM qui inclut V_i
12 $X' \leftarrow X' \cup (E_i \backslash V_i)$
13 Soient $X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}$ les composantes connexes de $G[X_i \backslash E_i]$
14 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$

premier cluster noté E_0. X' notera l'ensemble des sommets déjà traités. Cet ensemble est donc initialisé avec le premier cluster E_0. Dans la suite, nous noterons $X_1, X_2, \dots X_k$ les composantes connexes du sous-graphe $G[X \backslash E_0]$ induit par la suppression dans G des sommets de E_0. Chacun de ces ensembles X_i est inséré dans une file F en attente de traitement. Ainsi, on tient compte de la topologie du graphe en identifiant les parties indépendantes du graphe du point de vue du cluster qui vient de se former. Pour chaque élément X_i supprimé de la file F, on notera V_i l'ensemble des sommets de X' qui sont adjacents à au moins un sommet de X_i. V_i est donc un ensemble de sommets séparant X_i des autres sommets de X'. La différence avec *Least-TD* se présente au niveau de la construction du nouveau cluster E_i. Considérons le sous-graphe de G induit par V_i et X_i, c'est-à-dire $G[V_i \cup X_i]$. Dans un premier temps, l'ensemble V_i est complété en rajoutant dans G les arêtes absentes entre chaque paire de sommets de V_i (ligne 8). L'étape suivante (ligne 9) consiste en une triangulation de $G[V_i \cup X_i]$ utilisant *Min-Fill* et construit un graphe triangulé G_i des sommets de $V_i \cup X_i$. Les cliques maximales de G_i sont ensuite (ligne 10) mémorisées dans l'ensemble CM. Le fait que l'ensemble V_i soit une clique (cf. ligne 8) garantit l'inclusion de V_i dans au moins une des cliques maximales mémorisées dans CM. En fait, cette clique va constituer le nouveau cluster noté E_i qui sera ajouté à l'ensemble des clusters déjà calculés tout en mettant à jour l'ensemble X'. À ce stade, et comme le faisait *Least-TD*, on recalcule les nouvelles composantes connexes issues de la suppression des sommets de E_i et on les ajoute dans la file. Puis, on réitère le processus jusqu'à ce que la file soit vide.

La preuve de la validité de *Min-Fill-MG* est semblable à celle de *Least-TD* (voir le théorème 1), excepté concernant la terminaison. Nous nous limitons donc à prouver l'arrêt de *Min-Fill-MG* :

Théorème 3 *Min-Fill-MG calcule les clusters d'une décomposition arborescente.*

Preuve : Pour prouver que *Min-Fill-MG* se termine, il suffit de montrer qu'à chaque passage dans la boucle de la ligne 5, au moins un sommet de X_i sera ajouté à X'. Comme on choisit à chaque étape une clique maximale issue de la triangulation de $G[V_i \cup X_i] = G_i$ et qui contient V_i, il suffit de démontrer que ce cluster incluera V_i strictement (donc que $V_i \subsetneq E_i$).

D'après le théorème de Dirac [9], tout graphe triangulé possède au moins deux sommets *simpliciaux* (sommets dont tous les voisins forment une clique) qui ne sont pas voisins si le graphe n'est pas complet. Ainsi, on sait qu'il existe deux sommets x et x' dans G_i qui sont simpliciaux. On a alors deux possibilités :

1. x ou $x' \in V_i$. Sans manque de généralité, considérons $x \in V_i$. Puisque $x \in V_i$, on sait que x possède au moins un voisin v dans X_i. Puisque x est simplicial, l'ensemble de ses voisins forme une clique. Or, comme V_i a été complété (cf. ligne 8), x possède comme voisins au moins $(V_i \backslash \{x\}) \cup \{v\}$ qui est une clique de G_i. Ainsi, $V_i \cup \{v\}$ est aussi une clique de G_i. Cette clique est nécessairement contenue dans une clique maximale de G_i et on a ainsi au moins une clique E_i de CM telle que $V_i \subsetneq E_i$.
2. Ni x, ni x' ne sont dans V_i. Nous démontrons le résultat par induction sur la taille de X_i. L'hypothèse est que pour $|X_i| = k \geq 2$ (car ni x, ni x' ne sont dans V_i et si $|X_i| = 1$, on se retrouve dans le cas (1)), il existe une clique de G_i incluant strictement V_i. Pour la base de l'induction, on a $|X_i| = 2$. Dans ce cas, comme $G[X_i]$ est connexe, nécessairement x et x' sont voisins avant triangulation. Par conséquent, G_i forme une clique car on ne peut avoir deux sommets de X_i non voisins et simpliciaux. On suppose maintenant que la proposition est vérifiée pour $|X_i| = k \geq 2$ et on prouve qu'elle est aussi vérifiée pour $|X_i| = k+1$. Considérons x simplicial. Si x inclut V_i dans son voisinage, $V_i \cup \{x\}$ est une clique de G_i et le résultat est vérifié. Si x n'inclut pas V_i dans son voisinage, le sous-graphe de G_i induit par la suppression de x possède k sommets et il est triangulé. Dans ce cas, par hypothèse d'induction, il contient une clique incluant strictement V_i. *A fortiori*, G_i inclut cette clique et le résultat est vérifié. Il faut noter que si x inclut une partie de V_i dans son voisinage, comme x inclut nécessairement au moins un autre sommet v de X_i dans son voisinage (sinon l'hypothèse de connexité de X_i avant triangulation de $G[V_i \cup X_i]$ ne serait pas vérifiée) et que x est simplicial, v sera également voisin de

ces sommets dans le sous-graphe de G_i induit par la suppression de x.

En conséquence, on a bien un élément E_i de CM tel que $V_i \subsetneq E_i$. □

Théorème 4 *La complexité en temps de l'algorithme Min-Fill-MG est* $O(n^2(n+e'))$.

Preuve : Les lignes 1-4 sont réalisables en temps linéaire, soit $O(n+e)$, puisque le coût de calcul des composantes connexes de $G[X \backslash E_0]$ est borné par $O(n+e)$. Nous analysons maintenant le coût de la boucle (ligne 5). Tout d'abord, notons qu'il y a nécessairement moins de n insertions dans la file F car à chaque passage dans la boucle, on est assuré qu'au moins, un nouveau sommet aura été rajouté dans X', et donc supprimé de l'ensemble des sommets n'ayant pas encore été traités. Nous analysons maintenant le coût de chaque traitement associé à l'ajout d'un nouveau cluster, dont nous donnons pour chacun, sa complexité globale.

- Ligne 6 : $O(n^2)$ pour la même raison que *Least-TD*.
- Ligne 7 : $O(n(n+e'))$ pour la même raison que *Least-TD*, mais par contre en considérant e' comme nombre d'arêtes.
- Ligne 8 : $O(n^3)$ avec une majoration abusive.
- Ligne 9 : cette étape calcule une triangulation en utilisant l'heuristique *Min-Fill* qui a une complexité de $O(n(n+e'))$, soit globalement $O(n^2(n+e'))$.
- Ligne 10 : le calcul des cliques maximales d'un graphe triangulé peut se faire en temps linéaire (voir [11]), soit $O(n+e')$ ici. Ainsi, globalement le coût sera donc en $O(n(n+e'))$.
- Ligne 11 : comme le nombre de clusters est majoré par n, et que le coût du test d'inclusion est de $O(n)$, le coût total de cette étape peut facilement être majoré par $O(n^2)$, soit globalement par $O(n^3)$.
- Pour les lignes 12, 13 et 14 et pour les mêmes raisons que *Least-TD*, on peut majorer respectivement par $O(n^2)$, $O(n(n+e'))$ et $O(n^2)$.

Finalement, la complexité temporelle de l'algorithme *Min-Fill-MG* est de $O(n^2(n+e'))$. □

5 Évaluation expérimentale

Dans cette section, nous comparons la largeur des décompositions arborescentes produites par *Min-Fill*, *Least-TD* et *Min-Fill-MG* entre elles, mais aussi à la largeur arborescente quand celle-ci est connue. Pour *Least-TD* et *Min-Fill-MG*, le choix du premier cluster consiste à calculer une clique maximale contenant le sommet de plus grand degré dans le graphe. Les trois méthodes de décomposition sont implémentées en C++ au sein de notre bibliothèque graphique. Les expérimentations ont été réalisées sur des serveurs lames sous Linux Ubuntu 14.04 dotés chacun de deux processeurs Intel Xeon E5-2609 à 2,4 GHz et de 32 Go de mémoire. Elles portent principalement sur 1 668 instances CSP issues de la compétition CSP de 2008 [3] auxquelles s'ajoutent quelques instances issues du dépôt UCI sur les Réseaux de Croyance et quelques instances du problème de coloration de graphes. Pour les deux derniers cas, il s'agit d'instances dont la largeur arborescente est connue [2]. A noter que parmi toutes ces instances, certaines correspondent en fait à des hypergraphes dont les décompositions arborescentes sont obtenues en travaillant sur leur 2-section.

Pour comparer la largeur des décompositions produites par *Min-Fill*, *Least-TD* et *Min-Fill-MG*, nous avons considéré 38 instances dont la largeur est connue. La table 1 présente les résultats obtenus pour quelques-unes d'entre elles. Le nombre d'instances considéré peut paraître faible, mais il faut se rappeler que déterminer la largeur arborescente d'un graphe quelconque est un problème NP-difficile. Les méthodes exactes ne sont vraiment opérationnelles que sur des graphes de petite taille. Une alternative consiste à procéder à un encadrement de la largeur arborescente par des bornes inférieures et supérieures mais, faute de disposer de bornes de qualité, cette solution n'aboutit que trop rarement à déterminer w.

Nous avons observé que *Min-Fill* et *Min-Fill-MG* trouvent une décomposition optimale pour 35 instances alors que *Least-TD* n'y parvient que pour 24 instances. Dans les cas où la décomposition n'est pas optimale, la largeur obtenue est souvent proche de l'optimum. Ceci illustre bien l'aptitude de ces méthodes heuristiques à calculer des décompositions de qualité en temps raisonnable. En effet, chaque décomposition est ici calculée en moins de 0,12 s.

À présent, nous comparons les différentes méthodes heuristiques de décomposition entre elles. Nous considérons pour cela 1 668 instances CSP issues de la compétition CSP de 2008. La table 2 fournit les largeurs des décompositions obtenues pour une sélection d'instances représentatives des différentes tendances observées. *Least-TD* produit des décompositions ayant une largeur inférieure ou égale à celles de *Min-Fill* pour 1 005 des 1 668 instances. Pour 772 de ces instances, *Least-TD* améliore la largeur des décompositions produites par *Min-Fill*. L'amélioration peut être très significative comme c'est le cas, par exemple, pour l'instance `bqwh-18-141-37_ext` avec une largeur de 54

3. Voir http ://www.cril.univ-artois.fr/CPAI08 pour plus de détails.

	Instances	n	e	w	*Min-Fill*	*Least-TD*	*Min-Fill-MG*
					w^+	w^+	w^+
(a)	alarm.net	37	65	4	4	4	4
	barley.net	48	126	7	7	10	7
	hepar2.net	70	158	6	6	6	6
	water.net	32	123	8	10	11	9
(b)	primes-30-20-3-1	100	98	3	3	3	3
	primes-30-40-2-1	100	82	2	2	3	2
	mps-red-markshare1	230	12 835	79	79	79	79
	mps-red-markshare2-1	330	35 385	149	149	149	149
(c)	david	87	406	13	13	14	13
	miles500	128	1170	22	23	29	23
	myciel3	11	20	5	5	5	5
	myciel4	23	71	10	11	11	11

TABLE 1 – Nombre de sommets et d'arêtes, largeur arborescente (optimum) et largeur des décompositions produites par *Min-Fill*, *Least-TD* et *Min-Fill-MG* pour des instances dont la largeur arborescente w est connue. Ces instances proviennent du dépôt UCI sur les Réseaux de Croyance (a), de la compétition CSP de 2008 (b) et du problème de coloration de graphes (c).

pour *Least-TD* contre 73 pour *Min-Fill*. Concernant *Min-Fill-MG*, les largeurs produites sont strictement meilleures que celles de *Min-Fill* pour 522 instances et de qualité égale pour 749. À nouveau, le gain peut être important. Par exemple, la largeur de la décomposition de l'instance `ii-8e2` est de 149 pour *Min-Fill-MG* contre 179 pour *Min-Fill*. Enfin, *Least-TD* obtient des largeurs strictement inférieures à celles de *Min-Fill-MG* pour 733 instances et égales pour 254 instances.

Concernant le temps d'exécution, *Least-TD* permet de calculer des décompositions plus rapidement qu'avec *Min-Fill*. En effet, pour 85 % (respectivement 98 %) des instances, la décomposition est calculée en moins d'une seconde (resp. une minute) par *Least-TD* contre 83 % (resp. 95 %) pour *Min-Fill*. Par contre, *Min-Fill-MG* requiert un temps d'exécution plus long que les deux premières méthodes avec 57 % (resp. 76 %). Ces résultats étaient prévisibles au regard de la complexité des algorithmes mais aussi dans la mesure où l'implémentation actuelle de *Min-Fill-MG* est assez rudimentaire.

6 Conclusion

Dans cet article, nous avons introduit deux nouvelles heuristiques *Least-TD* et *Min-Fill-MG* pour le calcul de décompositions arborescentes. L'objectif était d'améliorer *Min-Fill*, la méthode de référence dans ce cadre. L'évaluation expérimentale nous a permis de montrer que nous améliorons la qualité des décompositions sur une majorité des instances, et souvent de manière significative. De plus, les temps de calculs sont, du moins pour *Least-TD*, améliorés. Ces deux nouvelles heuristiques servent ainsi à élargir profitablement le catalogue d'algorithmes de décomposition opérationnels sur le plan pratique, celui-ci étant limité depuis de nombreuses années à la seule heuristique *Min-Fill*.

Au niveau des perspectives, il semble rester beaucoup à faire. Déja pour des améliorations de la mise en œuvre de ces deux algorithmes, en particulier pour *Min-Fill-MG* qui semble significativement optimisable au niveau des temps de calcul. Au-delà, ces deux approches ouvrent sur un champ d'investigation prometteur, en particulier concernant *Least-TD*. En effet, le schéma d'algorithme utilisé doit permettre la mise en œuvre d'heuristiques différentes en proposant d'autres choix pour la sélection associée à la construction d'un nouveau cluster. En effet, nous avons arrêté ce choix ici à une unique possibilité alors même que d'autres sont envisageables comme par exemple choisir un ensemble de sommets de X_i qui accroîtrait le nombre de sommets de V_i directement supprimés plutôt que de minimiser le nombre de sommets de X_i ayant un voisin commun dans V_i comme *Least-TD* le fait. Enfin, il reste aussi à analyser ces nouvelles décompositions au regard de leur apport pour la résolution de CSP, pour le problème de décision, mais plus particulièrement, pour les problèmes de comptage et d'optimisation, et éventuellement aussi, étudier leur impact dans le cas de la compilation.

Références

[1] S. Arnborg, D. Corneil, and A. Proskuroswki. Complexity of finding embeddings in a k-tree. *SIAM Journal of Disc. Math.*, 8 :277–284, 1987.

Instances	n	e	*Min-Fill*		*Least-TD*		*Min-Fill MG*	
			tps	w^+	tps	w^+	tps	w^+
composed-25-10-20-5_ext	105	620	< 0,01	**24**	< 0,01	28	0,02	**24**
graph14-f28	916	4 638	0,45	239	0,48	**229**	119,47	230
par-8-2	700	2 800	0,04	47	0,06	**41**	1,32	43
par-16-4-c	648	3 723	0,05	**43**	0,08	83	1,11	**43**
radar-10-20-4.5-0.95-98	906	10 211	0,20	**112**	0,6	121	19,70	114
s4-4-3-6	624	9 152	0,43	192	0,27	**177**	147,38	187
tsp-25-3_ext	76	400	< 0,01	**25**	< 0,01	**25**	< 0,01	**25**
bf-0432-007	2 080	7 473	0,79	145	6,47	141	32,20	**132**
ii-8e2	1 740	10 785	1,70	179	11,04	163	178,71	**149**
js-taillard-20-15-95-2	300	3 130	0,06	117	0,04	**109**	3,66	110
4-insertions-4-5	475	1 795	0,07	94	0,14	**81**	7,67	88
fapp04-0300-1	300	1 799	0,07	134	0,03	**123**	30,82	131
bqwh-18-141-37_ext	141	883	0,01	73	< 0,01	**54**	0,43	69
graph4	400	2 244	0,05	100	0,05	109	3,71	**97**
ssa-0432-003	870	2 022	0,12	34	0,46	65	2,00	**33**
games120-7	120	638	< 0,01	39	< 0,01	43	0,14	**36**

TABLE 2 – Nombre de sommets et d'arêtes, largeur des décompositions produites par *Min-Fill*, *Least-TD* et *Min-Fill-MG* pour des instances CSP de la compétition de solveurs de 2008. Les meilleures largeurs obtenues pour chaque instance sont en gras.

[2] J. Berg and M. Järvisalo. Sat-based approaches to treewidth computation : An evaluation. In *Proceedings of ICTAI*, pages 328–335, 2014.

[3] C. Berge. *Graphs and Hypergraphs.* Elsevier, 1973.

[4] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs : Basic properties and comparison. *Constraints*, 4(3) :199–240, 1999.

[5] C. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio Link Frequency Assignment. *Constraints*, 4 :79–89, 1999.

[6] R. Dechter. *Constraint processing.* Morgan Kaufmann Publishers, 2003.

[7] R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125 :93–118, 2001.

[8] R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38 :353–366, 1989.

[9] G.A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg 25*, MR24 -57 :71–76, 1961.

[10] H. Fargier and M.-C. Vilarem. Compiling CSPs into tree-driven automata for interactive solving. *Constraints*, 9(4) :263–287, 2004.

[11] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[12] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146 :43–75, 2003.

[13] P. Jégou and C. Terrioux. Tree-decompositions with connected clusters for solving constraint networks. In *Proceedings of CP*, pages 407–423, 2014.

[14] P. Jégou and C. Terrioux. Un nouveau paramètre de graphes pour la résolution de CSP par décomposition. In *Actes des JFPC*, pages 77–88, 2014.

[15] U. Kjaerulff. Triangulation of graphs - algorithms giving small total state space. Technical report, Judex R.R. Aalborg, Denmark, 1990.

[16] N. Robertson and P.D. Seymour. Graph minors II : Algorithmic aspects of treewidth. *Algorithms*, 7 :309–322, 1986.

[17] D. J. Rose. A graph theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.

[18] S. Subbarayan. An empirical comparison of csp decomposition methods. In *Proceedings of the CP Doctoral Program*, 2007.

[19] R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13 (3) :566–579, 1984.

[20] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3) :410–421, 1979.

Compilation de réseaux de contraintes en graphes de décision décomposables multivalués

Frédéric Koriche Jean-Marie Lagniez Pierre Marquis Samuel Thomas

CRIL-CNRS, Université d'Artois, Lens, France

{koriche,lagniez,marquis,thomas}@cril.univ-artois.fr

Résumé

Dans cet article, nous présentons et évaluons un algorithme de compilation de réseaux de contraintes (CNs) en graphes de décision décomposables multivalués (MDDG). Bien que le langage MDDG contienne le langage Decision-DNNF comme sous-ensemble propre, il offre les mêmes requêtes et les mêmes transformations réalisables en temps polynomial, ce qui en fait un langage intéressant pour beaucoup d'applications. Notre large panel d'expérimentations a montré que le compilateur cn2mddg que nous avons développé parvient à compiler des CNs qui se trouvent généralement hors de portée des approches standard basées sur la transformation du réseau en CNF, compilée ensuite en Decision-DNNF. De plus, la taille de la forme compilée obtenue se révèle plus concise, parfois de plusieurs ordres de grandeur.

Abstract

We present and evaluate a top-down algorithm for compiling finite-domain constraint networks (CNs) into the language MDDG of multivalued decomposable decision graphs. Though it includes Decision-DNNF as a proper subset, MDDGoffers the same key tractable queries and transformations as Decision-DNNF, which makes it useful for many applications. Intensive experiments showed that our compiler cn2mddg succeeds in compiling CNs which are out of the reach of standard approaches based on a translation of the input network to CNF, followed by a compilation to Decision-DNNF. Furthermore, the sizes of the resulting compiled representations turn out to be much smaller (sometimes by several orders of magnitude).

1 Introduction

La programmation par contraintes (CP) est depuis longtemps reconnue comme un paradigme de choix pour la représentation et la résolution de problèmes combinatoires [22]. Le problème est représenté de manière compacte et intuitive en utilisant un réseau de contraintes (CN) ; celui-ci consiste en un ensemble de variables, chacune étant associée à un domaine de valeurs, et un ensemble de contraintes qui interconnectent les variables en spécifiant, d'une manière ou d'une autre, leurs n-uplets de valeurs autorisés. Malgré le succès indéniable de cette approche déclarative, un des défis majeurs de la programmation par contraintes est d'offrir des garanties de performance pour répondre aux requêtes posées par l'utilisateur, qui se résument souvent à résoudre des (instances de) problèmes NP-difficiles. Comme il est souligné dans [13], le critère de garantie de performance est crucial pour les applications en ligne, telles que les logiciels de configuration [17] et les systèmes de recommandation [6], où les requêtes posées « à la volée » par l'utilisateur doivent être résolues en temps réel.

Le but de cet article est de répondre à ce critère en utilisant la compilation de connaissances [8]. L'idée générale est de convertir un langage de contraintes vers un langage de compilation cible permettant de résoudre diverses tâches de calcul (que l'on classe souvent en requêtes et transformations) en temps polynomial. Bien que beaucoup de requêtes soient intraitables lorsque le problème est formulé en CN, elles deviennent traitables à partir d'une forme compilée de celui-ci, assurant ainsi une garantie de performance en ligne quand la forme compilée est de taille raisonnable.

Nous présentons un algorithme descendant cn2mddg pour la compilation de CNs, à domaines finis, en graphes de décision décomposables multivalués. cn2mddg reçoit en entrée un CN représenté sous le format XCSP 2.1 [23]. Notre algorithme de compilation retourne en sortie une représentation des solutions du CN dans le langage MDDG (graphes de décision décomposables multivalués). MDDG est l'extension vers les domaines non booléens du langage DDG [11], aussi connu sous le nom de Decision-DNNF [21]. Il est basé sur

des nœuds $\wedge$ décomposables et des nœuds de décision (multivalués). Comme le langage des Decision-DNNF, le langage des MDDG permet le traitement polynomial de plusieurs classes de requêtes, telles que la recherche d'une solution, le comptage de solutions (possiblement pondérées), l'énumération des solutions (avec un délai polynomial), et l'optimisation sous fonction objectif linéaire. Ce langage autorise aussi plusieurs transformations polynomiales, telles que le conditionnement, c'est-à-dire l'instanciation de variables, et plus généralement l'ajout de contraintes unaires.

cn2mddg exploite une technique de cache spécifique, ainsi qu'une nouvelle heuristique de choix de variable fondée sur une notion de centralité considérée en algorithmique des graphes (*betweenness centrality*), et détecte les contraintes universelles pendant la recherche afin d'effectuer des simplifications supplémentaires. cn2mddg utilise une approche *directe* : le CN donné en entrée est compilé directement en MDDG, sans passer par une formule intermédiaire. A l'opposé, les compilateurs standard comme c2d [7] ou Dsharp [20], nécessitent une approche *indirecte* pour prendre en charge les réseaux de contraintes : le CN donné est d'abord traduit en une CNF équivalente, qui est ensuite transformée par le compilateur en Decision-DNNF.

Nous avons expérimenté notre compilateur cn2mddg sur un grand nombre de jeux d'essai (173) provenant de différentes familles (15). A la lumière des résultats expérimentaux, il apparaît que les approches indirectes sont, dans la majorité des cas pratiques, inutilisables : quel que soit l'encodage utilisé, la traduction du CN en formule clausale (CNF) équivalente requiert un grand nombre de variables booléennes, et induit une perte de structure, inhérente à la représentation clausale (comparée à la représentation graphique d'un réseau de contraintes). En compilant directement le CN sans passer par une formule booléenne intermédiaire, notre compilateur cn2mddg est bien plus robuste puisqu'il réussit à compiler beaucoup de CNs qui se sont révélés hors de portée des approches indirectes, utilisant une traduction préliminaire en formule booléenne. De plus, les tailles des formes compilées obtenues sont plus concises, le gain de taille étant parfois réduit de plusieurs ordres de grandeur.

Dans la suite de l'article, nous commençons par introduire quelques notions de base, portant sur les réseaux de contraintes, puis nous présentons le langage MDDG. Nous décrivons ensuite le compilateur cn2mddg, et concluons en présentant quelques résultats obtenus lors de nos expérimentations. Notre compilateur, ainsi que les traducteurs, les jeux d'essai utilisés pour nos expérimentations et des résultats suplémentaires, peuvent être téléchargés à partir de l'adresse suivante : http://www.cril.fr/KC/.

2 Préliminaires

Rappelons qu'un réseau de contraintes (CN) (fini) est un triplet $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, consistant en un ensemble de variables $\mathcal{X} = \{X_1, \ldots, X_n\}$, un ensemble de domaines $\mathcal{D} = \{D_1, \ldots, D_n\}$, et un ensemble de contraintes $\mathcal{C} = \{C_1, \ldots, C_m\}$. Chaque domaine D_i est un ensemble fini contenant les valeurs possibles de la variable X_i. Chaque contrainte C_j pose une restriction sur les combinaisons de valeurs d'un sous-ensemble de variables. Formellement, $C_j = (S_j, R_j)$, où $S_j = \{X_{j1}, \ldots, X_{jk}\}$ est une partie de X, appelée *portée* (ou *scope*) de C_j, et R_j est un prédicat sur le produit cartésien de $D_{j1} \times \ldots \times D_{jk}$, appelé *relation* de C_j. R_j peut être représenté en extension par la liste des n-uplets autorisés (ou la liste des n-uplets interdits), ou en intension via un oracle, c'est-à-dire une fonction $D_{j1} \times \ldots \times D_{jk}$ vers $\{0, 1\}$ calculable en temps polynomial en la taille de l'entrée. L'*arité* d'une contrainte est donnée par le cardinal de sa portée. Les contraintes d'arité 2 sont dites *binaires* et celles d'arité supérieure à 2 sont dites n-aires.

Exemple 1 *Dans la suite, nous considérons le réseau* CN $\mathcal{N}$ *défini sur quatre variables* X_1, X_2, X_3 *et* X_4*, chacune étant associée au même domaine* $\{0, 1, 2\}$*, par les trois contraintes* C_1, C_2 *et* C_3*, caractérisées par les expressions suivantes :*

- $C_1 = (X_1 \neq X_2)$;
- $C_2 = (X_2 = 0) \vee (X_2 = 1) \vee (X_2 = X_3 + X_4 + 1)$;
- $C_3 = (X_3 > X_4)$.

Soit S un sous-ensemble de variables de $\mathcal{X}$. Nous appelons *état* (de décision) sur S, une fonction $\boldsymbol{s}$ qui associe à chaque variable X_i de S un sous-ensemble $\boldsymbol{s}(X_i)$ de valeurs dans D_i. Dans la suite, les états sont aussi notés comme l'union d'affectations élémentaires, c'est-à-dire des ensembles de la forme $\{\langle X_i, x_j \rangle\}$, où $x_j \in \boldsymbol{s}(X_i)$. *scope*$(\boldsymbol{s})$ représente l'ensemble S des variables pour lesquelles $\boldsymbol{s}$ est défini. Un état $\boldsymbol{s}$ est dit *partiel* si *scope*$(\boldsymbol{s})$ est un sous-ensemble strict de $\mathcal{X}$, sinon $\boldsymbol{s}$ est dit *complet*. Une variable X_i de *scope*$(\boldsymbol{s})$ est dite *instanciée* si $\boldsymbol{s}(X_i)$ est un singleton. L'ensemble des variables instanciées de $\boldsymbol{s}$ est noté *single*$(\boldsymbol{s})$. Un état $\boldsymbol{s}$ est dit instancié lorsque toutes ses variables sont instanciées, i.e. *scope*$(\boldsymbol{s})$ = *single*$(\boldsymbol{s})$.

Pour un état $\boldsymbol{s}$ et un sous-ensemble de variables $T \subseteq$ *scope*$(\boldsymbol{s})$, nous notons $\boldsymbol{s}[T]$ la restriction de $\boldsymbol{s}$ à T, autrement dit $\boldsymbol{s}[T]$ est l'ensemble $\{\langle X_i, x_j \rangle \in \boldsymbol{s} \mid X_i \in T\}$. Une instanciation $\boldsymbol{s}$ satisfait une contrainte $C_j = (S_j, R_j)$ si $S_j \subseteq$ *scope*$(\boldsymbol{s})$ et $R_j(x_{j1}, \ldots, x_{jk}) = 1$, pour tout $l \in 1, \ldots, k$, tel que $\langle X_{jl}, x_{jl} \rangle \in \boldsymbol{s}[S_j]$. Une *solution* d'un CN $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est une instanciation complète $\boldsymbol{s}$ satisfaisant toutes les contraintes C_j de $\mathcal{C}$. Par exemple, $\boldsymbol{s} = \{\langle X_1, 1 \rangle, \langle X_2, 0 \rangle, \langle X_4, 1 \rangle, \langle X_4, 0 \rangle\}$ est une solution du CN de l'exemple 1.

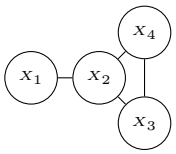


FIGURE 1 – Graphe primal du CN de l'exemple 1.

Soit un CN $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ et un état $\boldsymbol{s}$ défini sur une partie de $\mathcal{X}$. Le *conditionnement* de $\mathcal{N}$ par $\boldsymbol{s}$, noté $\mathcal{N} \mid \boldsymbol{s}$, est le CN $(\mathcal{X}', \mathcal{D}', \mathcal{C}')$ défini par les conditions suivantes : $\mathcal{X}' = \mathcal{X} \setminus \mathit{single}(\boldsymbol{s})$; à chaque domaine D_i de $\mathcal{D}$, on associe le domaine $D'_i \in \mathcal{D}'$, où $D'_i = D_i$ si $X_i \notin \mathit{scope}(\boldsymbol{s})$ et $D'_i = s(D_i)$ sinon ; enfin, à chaque contrainte $C_j = (S_j, R_j)$ de $\mathcal{C}$, on associe la contrainte $C'_j = (S'_j, R'_j)$, où $S'_j = S_j \setminus \mathit{single}(\boldsymbol{s})$ et R'_j est la restriction de R_j à S'_j.

Le graphe primal d'un CN $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est le graphe non orienté G donné par l'ensemble des sommets $\mathcal{X}$ et l'ensemble des arêtes $\mathcal{E}$, tel que $\{X_p, X_q\} \in \mathcal{E}$ si et seulement si $\{X_p, X_q\}$ est un sous-ensemble de la portée S_j d'une contrainte C_j de $\mathcal{C}$. Par exemple, le graphe primal de l'exemple 1 est décrit dans la figure 1.

3 Le langage MDDG

Dans un premier temps, nous présentons le langage des graphes de décision multivalués (MDG) :

Définition 1 *(MDG) Soit un ensemble fini $\mathcal{X}$ de variables à domaine fini. Le langage MDG des graphes de décision multivalués (à lecture unique) sur $\mathcal{X}$ est l'ensemble des graphes Δ orientés sans circuit à racine unique, tels que chaque feuille est étiquetée par $\top$ (vrai) ou $\bot$ (faux), et chaque nœud interne N est soit un nœud $\wedge$, soit $N = \wedge(N_1, \ldots, N_i)$, soit un nœud de décision N associé à une variable $X_i \in \mathcal{X}$, autrement dit un nœud $\vee$ déterministe $N = \vee(N_1, \ldots, N_j)$ tel que $D_i = \{x_{i1}, \ldots, x_{ij}\}$ et l'arc entre N et $N_k (k \in 1, \ldots, j)$ est étiqueté par l'affectation élémentaire $\{\langle X_i, x_{ik}\rangle\}$. Les chemins dans Δ doivent satisfaire la propriété de « lecture unique » (*read-once*) : pour tout chemin depuis la racine de Δ jusqu'à une feuille, et pour toute variable $X_i \in \mathcal{X}$, au maximum un seul arc peut être étiqueté par une affectation élémentaire portant sur X_i.*

Pour chaque nœud N d'un graphe Δ de MDG, l'ensemble des variables de N, noté $\mathit{Var}(N)$, est inductivement défini par les conditions suivantes :

— si N est une feuille, alors $\mathit{Var}(N) = \varnothing$;
— si $N = \wedge(N_1, \ldots, N_i)$ est un nœud $\wedge$, alors $\mathit{Var}(N) = \bigcup_{k=1}^{i} \mathit{Var}(N_k)$;

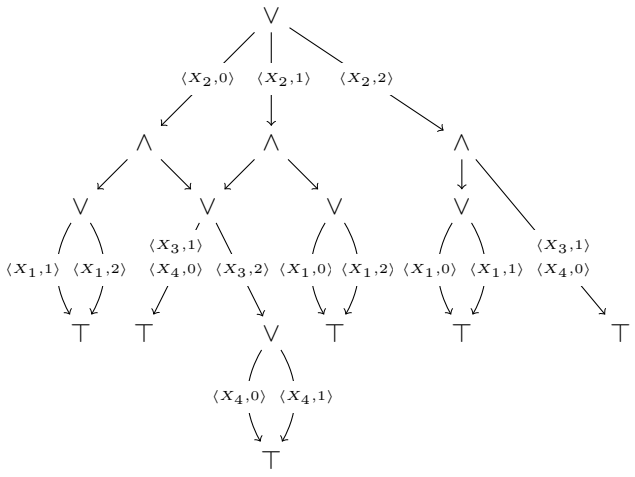


FIGURE 2 – Un MDDG équivalent au CN de l'exemple 1.

— si N est un nœud de décision $N = \vee(N_1, \ldots, N_j)$ associé à une variable X, alors $\mathit{Var}(N) = \{X\} \cup \bigcup_{k=1}^{j} \mathit{Var}(N_k)$.

Soit $\boldsymbol{s}$ une instanciation complète sur $\mathcal{X}$ et soit Δ un graphe de MDG sur $\mathcal{X}$, ayant pour racine N. Le graphe $\mathit{eval}(N, \boldsymbol{s})$ est défini inductivement par les conditions suivantes :

— si N est une feuille, alors $\mathit{eval}(N, \boldsymbol{s}) = N$;
— si $N = \wedge(N_1, \ldots, N_i)$ est un nœud $\wedge$, alors $\mathit{eval}(N, \boldsymbol{s}) = \wedge(\mathit{eval}(N_1, \boldsymbol{s}), \ldots, \mathit{eval}(N_i, \boldsymbol{s}))$;
— si N est un nœud de décision $N = \vee(N_1, \ldots, N_j)$ associé à la variable X, alors $\mathit{eval}(N, \boldsymbol{s}) = \mathit{eval}(N_k, \boldsymbol{s})$ avec $\langle X, x_k\rangle \in \boldsymbol{s}$.

$\boldsymbol{s}$ est une *solution* de Δ si et seulement si le graphe de MDG sans noeud de décision $\mathit{eval}(N, \boldsymbol{s})$ est évalué à vrai.

Le langage MDDG examiné dans cette étude est un sous-ensemble des MDG contenant des nœuds *décomposables*, c'est-à-dire des nœuds $\wedge$ dont les fils ne partagent aucune variable en commun.

Définition 2 *(MDDG) Soit un ensemble fini $\mathcal{X}$ de variables à domaine fini, le langage MDDG sur $\mathcal{X}$ est le sous-ensemble des représentations en MDG Δ où tout nœud $N = \wedge(N_1, \ldots, N_i)$ de type $\wedge$ est décomposable, c'est-à-dire $\mathit{Var}(N_k) \cap \mathit{Var}(N_l) = \varnothing$ pour tous $k, l \in \{1, \ldots, i\}$, tels que $k \neq l$.*

A titre d'illustration, le graphe MDDG décrit dans la figure 2 est équivalent au CN donné dans l'exemple 1 : les deux représentations ont le même ensemble de solutions. Pour des raisons de lisibilité, les arcs conduisant à une feuille $\bot$ ne sont pas représentés sur la figure et la feuille $\top$ est dupliquée. De plus, les nœuds avec un seul fils sont supprimés et leurs étiquettes (dans le cas

des nœuds de décision) sont regroupées avec celles de leur arc incident. Les affectation élémentaires correspondantes sont typiquement obtenues par propagation (dans ce cas, les nœuds $\vee$ ne sont pas explicitement construits, nous les mentionnons dans la définition du langage MDDG dans un souci de simplicité).

Le langage des Decision-DNNF [21, 11] est un sous-ensemble du langage des MDDG où chaque variable possède un domaine booléen. En plus du gain d'expressivité obtenu en autorisant des domaines non booléens, les MDDG peuvent essentiellement traiter les mêmes requêtes et les mêmes transformations que celles des Decision-DNNF (comptage de solutions pondéré, énumération de solutions, optimisation pour une fonction objectif linéaire et conditionnement). Les algorithmes en temps polynomial utilisés pour résoudre ces requêtes et transformations dans les Decision-DNNF peuvent être étendus aux MDDG, de façon triviale.

MDDG possède aussi des similarités avec le langage MDD considéré dans [2], ainsi qu'avec le langage AOMDD considéré dans [18, 19], mais ne coïncide avec aucun d'entre eux. Ainsi, MDD et MDDG ne sont pas comparables en terme d'inclusion. D'un côté, une représentation en MDD est une structure non déterministe (plus d'un arc sortant d'un nœud de décision étiqueté par une variable X_i peut être étiqueté par une même valeur du domaine D_i de X_i), alors que les nœuds de décision des MDDG sont toujours déterministes. D'un autre côté, les représentations en MDDG autorisent des nœuds de type $\wedge$, alors que les nœuds internes des MDD sont tous des nœuds de décision. De la même manière, AOMDD et MDDG ne sont pas comparables du point de vue de l'inclusion. AOMDD est adapté à la compilation de modèles graphiques (en particulier, des réseaux bayésiens), et permet la représentation de fonctions non booléennes (comme des fonctions d'utilité ou de coût, des distributions de probabilités discrètes, etc.), alors que le langage MDDG ne le permet pas. De plus, les AOMDD sont des structures ordonnées qui respectent un pseudo-arbre induit par un ordre prédéfini d'élimination des variables - c'est un prérequis indispensable pour assurer la canonicité. A l'opposé, les MDDG n'imposent pas une telle condition (les représentations en MDDG ne sont pas canoniques).

4 Un compilateur MDDG de type descendant

Nous avons développé un compilateur cn2mddg de type descendant qui prend en entrée un CN représenté sous le format XCSP 2.1 [23], et retourne une représentation MDDG équivalente à celui-ci, c'est-à-dire ayant les mêmes solutions. Toutes les caractéristiques basiques offertes par le format XCSP 2.1 sont prises en compte. En particulier, les contraintes peuvent être représentées en intension (en tant que « prédicats ») ou en extension (en tant que « relations »). Cependant, seules trois contraintes globales sont supportées par notre compilateur dans sa forme actuelle : *allDifferent*, *weightedSum* (c'est-à-dire les contraintes linéaires), et *element*.[1]

L'architecture de notre compilateur est similaire à celle des compilateurs de type descendant pour les domaines booléens, comme c2d (http://reasoning.cs.ucla.edu/c2d/), ou Dsharp (http://www.haz.ca/research/dsharp/), tout deux ayant pour cible le langage Decision-DNNF. Notre compilateur est fondé sur un algorithme de recherche de solutions, suivant la trace d'un *solver* [15]. Il utilise des techniques similaires à celles proposées dans c2d et dans Dsharp, dont l'analyse de conflits pour guider la recherche, la propagation de contraintes afin de simplifier la formule, la mise en cache (*caching*) de composantes afin d'éviter la duplication de parties identiques pour la forme compilée, et une heuristique de choix de variable dynamique (comme Dsharp qui tire parti de l'heuristique de choix de variable *vsads*).[2] La méthode de *caching* et l'heuristique de choix de variable utilisées dans cn2mddg sont spécifiques à la nature du problème donné en entrée (un CN), qui est plus structuré que le format « plat » des CNF. De plus, notre algorithme exploite une méthode spécifique pour gérer les contraintes universelles, permettant d'effectuer plus de simplifications lors de la compilation.

Les contraintes universelles. Les contraintes universelles sont des contraintes qui sont nécessairement satisfaites, quelles que soient les affectations des variables apparaîssant dans leur portée (suivant le domaine courant des variables). Par exemple, avec le CN donné dans l'exemple 1, quand la contrainte C_2 est conditionnée par une affectation élémentaire $\{\langle X_2, 0\rangle\}$ ou $\{\langle X_2, 1\rangle\}$, C_2 devient universelle. A chaque étape de la compilation, les contraintes universelles sont détectées. L'universalité de chaque contrainte $C_j = (S_j, R_j) \in \mathcal{C}$ pour laquelle il existe au moins un $X_i \in s_j$ tel que D_i a été réduit par propagation à la suite de la dernière instanciation élémentaire est testée. Nous cherchons une instanciation $\boldsymbol{s}$ des variables dans la portée de C_j parmi les valeurs de leur domaine courant tel que $\boldsymbol{s}$ ne satisfait pas R_j. C_j est valide si et seulement si nous ne pouvons pas trouver une telle instanciation $\boldsymbol{s}$. Pour des raisons d'efficacité, $\boldsymbol{s}$ est cherchée d'une façon paresseuse. Une fois une instancia-

1. Ces contraintes peuvent aussi être encodées comme des « relations », mais dans ce cas il ne sera pas possible de profiter des propagateurs dédiés.

2. Dans c2d l'heuristique de choix de variable est statique.

tion $\boldsymbol{s}$ trouvée, elle est sauvegardée, puis reconsidérée en priorité lorsque l'universalité de C_j est à nouveau testée. Une fois détectée, une contrainte universelle C_j est simplement supprimée du réseau courant. Cela simplifie évidemment la suite des traitements (pas besoin de prendre en compte ces contraintes), favorise la décomposition (dû à la suppression d'arcs dans le graphe primal du CN), et influence l'heuristique de choix de variable. Pour les compilateurs de type Decision-DNNF, la gestion des contraintes universelles revient simplement à ignorer toutes les clauses partageant un littéral avec l'interprétation partielle courante.

Le caching. La mise en cache (*caching*) est une technique cruciale pour les compilateurs calculant des représentations à base de graphes orientés sans circuit (DAG). Le but est d'éviter d'explorer le même sous-problème deux fois ou plus, et de dupliquer la partie compilée. En effet, du fait de l'interchangeabilitée (conditionnelle) des valeurs dans beaucoup de réseaux, il arrive souvent que deux états de décision distincts $\boldsymbol{s}_1$ et $\boldsymbol{s}_2$ considérés successivement pendant la recherche conduisent au même sous-problème, autrement dit $\mathcal{N} \mid \boldsymbol{s}_1$ et $\mathcal{N} \mid \boldsymbol{s}_2$ sont équivalents. Dans une telle situation, plutôt que de compiler les deux réseaux, il est plus judicieux de ne compiler que $\mathcal{N} \mid \boldsymbol{s}_1$, puis d'enregistrer dans le cache une clé de $\mathcal{N} \mid \boldsymbol{s}_1$ associée à la racine N du nœud de sa représentation en MDDG et détecter que $\mathcal{N} \mid \boldsymbol{s}_2$ est équivalent à $\mathcal{N} \mid \boldsymbol{s}_1$ en cherchant dans le cache à chaque étape de la compilation. Ainsi, au lieu de calculer le graphe correspondant à $\mathcal{N} \mid \boldsymbol{s}_2$, il suffit de créer un arc pointant vers N. Par exemple, pour le CN donné dans l'exemple 1, la composante sur $\{X_3, X_4\}$ obtenue par décomposition dynamique est la même pour les états $\{\langle X_2, 0\rangle\}$ et $\{\langle X_2, 1\rangle\}$; nous n'avons donc pas besoin de la dupliquer.

Cependant, tester l'équivalence de deux CNs est un problème coNP-complet en général, ce problème devant être considéré un nombre exponentiel de fois durant la compilation. Pour ces raisons, il est impossible de faire du *caching* en force brute, où l'ensemble des réseaux différents $\mathcal{N} \mid \boldsymbol{s}$ rencontrés lors de la recherche seraient considérés (cela demanderait un temps de compilation ingérable). De ce fait, deux réseaux $\mathcal{N} \mid \boldsymbol{s}_1$ et $\mathcal{N} \mid \boldsymbol{s}_2$ sont considérés comme équivalents lorsqu'ils sont identiques.

La principale difficulté du *caching* concerne la taille des entrées du cache, qui doivent rester aussi petites que possible. Pour notre mise en cache, nous enregistrons d'abord les domaines courants des variables du problème, c'est-à-dire $\boldsymbol{s}$ restreint à ses variables non affectées. Stocker l'ensemble des contraintes courantes serait bien trop gourmand en espace. Heureusement, c'est inutile dans la plupart des cas. En effet, toute contrainte $C_j = (S_j, R_j)$ telle que $S_j \cap \mathit{single}(\boldsymbol{s}) = \varnothing$ n'a pas besoin d'être sauvegardée (étant donné que qu'elle n'est pas affectée). De plus, aucune contrainte $C_j = (S_j, R_j)$ binaire du réseau donné en entrée n'a besoin d'être enregistrée à partir du moment où ce réseau est arc-cohérent. En effet, si aucune variable de $S_j = \{X_i, X_k\}$ n'a été instanciée, alors nous sommes dans le cas précédent. Si les deux variables X_i, X_k de S_j sont instanciées alors soit la contrainte est universelle, soit elle est incohérente et donc, il est inutile de la stocker dans aucun des deux cas. Enfin, si une seule variable X_i de S_j est instanciée (disons à la valeur x_i) alors la projection sur la variable restante (non instanciée) X_k de la restriction R_j à $X_i = x_i$, coïncide avec la restriction de $\boldsymbol{s}$ à $\{X_k\}$ quand le réseau courant est arc-cohérent. [3] De manière similaire, nous n'avons pas besoin d'enregistrer les contraintes *allDifferent* qui peuvent être considérées comme des conjonctions de contraintes binaires. Les contraintes restantes sont sauvegardées dans notre cache. Pour les contraintes représentées en intension, les variables instanciées dans $\boldsymbol{s}$ sont remplacées par leurs valeurs dans les prédicats, et on effectue une étape de simplification dans le but de réduire les représentations des contraintes. Les contraintes représentées en extension sont stockées explicitement. Finalement, pour chaque contrainte *weightedSum*, les variables instanciées dans $\boldsymbol{s}$ sont remplacées par leurs valeurs, les contraintes sont simplifiées et seul le terme constant en résultant a besoin d'être sauvegardé. Les contraintes de type *element* qui sont binaires n'ont pas besoin d'être sauvegardées ; pour celles qui sont n-aires, on enregistre en cache $\{\langle X_i, x_i\rangle \in \boldsymbol{s} \mid X_i \in S_j\}$.

L'heuristique de choix de variable. Notre heuristique de choix de variable *bc* est basée sur une notion de centralité (*betweenness centrality*) utilisée en algorithmique des graphes [5]. Étant donné un nœud X_i dans un graphe (dans notre cas, le graphe primal du CN courant dans lequel les nœuds sont identifiés par les variables qu'ils étiquètent), $bc(X_i)$ est donné par le nombre de plus courts chemins entre toute paire de nœuds passant par X_i. Formellement,

$$bc(X_i) = \Sigma_{X_j \neq X_i \neq X_k} \frac{\sigma_{X_i}(X_j, X_k)}{\sigma(X_j, X_k)}$$

où X_i, X_j et X_k sont des nœuds du réseau, $\sigma(X_j, X_k)$ est le nombre de plus courts chemins de X_j à X_k et $\sigma_{X_i}(X_j, X_k)$ est le nombre de ces chemins passant par

3. Cela rappelle le traitement des clauses binaires dans Dsharp, qui n'ont pas besoin de figurer dans le cache du moment où la propagation unitaire a été effectuée [20].

Algorithm 1: cn2mddg($\mathcal{N}$)

entrée: un réseau de contraintes $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, et un état de décision $\mathtt{s}$ sur une partie de $\mathcal{X}$
sortie : la racine N d'une représentation MDDG équivalente à $\mathcal{N} \mid \mathtt{s}$

```
1  s ← ac(N, s)
2  N ← N | s
3  if unsat(N) then return leaf(⊥)
4  if #(X) = 0 then return leaf(⊤)
5  if cache(N) ≠ nil then return cache(N)
6  C ← removeUniversal(C)
7  CoCo ← connectedComponents(N)
8  N∧ ← ∅
9  foreach Co ∈ CoCo do
10     Xi ← selectVariable(Co)
11     N∨ ← ∅
12     foreach xi s.t. ⟨Xi, xi⟩ ∈ s do
13        N∨ ←
          N∨ ∪ cn2mddg(N, s[Co \ {Xi}] ∪ {⟨Xi, xi⟩})
14     N∧ ← N∧ ∪ dNode(Xi, N∨)
15 N ← aNode(N∧)
16 cache(N) ← N
17 return N
```

X_i. Dans le CN de l'exemple 1, X_2 est l'unique variable maximisant la valeur de *bc*. Il apparaît clairement qu'affecter en priorité les variables centrales du graphe primal $(\mathcal{X}, \mathcal{E})$ d'un CN favorise la génération de composantes connexes disjointes de taille similaire. Ceci permet la décomposition du réseau en plusieurs sous-réseaux indépendants (portant sur des ensembles disjoints de variables), de taille proche, pouvant être compilés séparément et rassemblés par un nœud $\wedge$ dans la représentation cible en MDDG. Le calcul de la centralité de tous les nœuds dans $(\mathcal{X}, \mathcal{E})$ peut être effectué en $\mathcal{O}(n.p)$, où $n = \#(\mathcal{X})$ et $p = \#(\mathcal{E})$. En pratique, le calcul de $bc(X_i)$ pour chaque nœud X_i du graphe $(\mathcal{X}, \mathcal{E})$ d'un CN est suffisamment efficace pour être effectué dynamiquement, c'est-à-dire être calculé pour chaque réseau rencontré lors de la compilation.

Le compilateur cn2mddg. L'algorithme 1 présente le pseudo-code du compilateur cn2mddg. La compilation d'un CN $\mathcal{N}$ s'effectue en appellant cn2mddg sur celui-ci et sur l'état de décision initial $\boldsymbol{s} = \{\langle X_i, x_i\rangle \mid X_i \in \mathcal{X}, x_i \in D_i\}$. Tout d'abord (ligne 1), on établit l'arc-cohérence de $\mathcal{N}$ avec $\boldsymbol{s}$ (les valeurs des variables apparaîssant dans $\boldsymbol{s}$ n'étant pas supportées par $\mathcal{N}$ sont supprimées de l'état). Pour des raisons d'efficacité, l'arc-cohérence est assurée au tout début (au premier appel) puis est maintenue dynamiquement à chaque fois qu'une nouvelle affectation élémentaire est considérée (ligne 13). Ensuite, $\mathcal{N}$ est conditionné par l'état courant (ligne 2). [4] A la ligne 3, nous appelons un *solver* CSP afin de déterminer si $\mathcal{N}$ est cohérent. Nous avons développé notre propre *solver*, basé sur une recherche en profondeur avec retour en arrière chronologique, utilisant l'heuristique dirigée par les conflits (devenue standard) *dom/wdeg* [4]. L'arc-cohérence est maintenue à chaque point de choix. Un poids est associé à chaque contrainte C_j de $\mathcal{C}$, et est incrémenté chaque fois qu'un conflit est détecté. Si $\mathcal{N}$ est incohérent, alors il est équivalent à la formule MDDG réduite à la feuille étiquetée par $\bot$, retournée par l'algorithme. La ligne 4 concerne le dernier cas de base, quand toutes les variables de $\mathcal{X}$ ont été considérées. Dans ce cas, $\mathcal{N}$ est équivalent à la formule MDDG $\top$, retournée par l'algorithme. On recherche si le réseau courant $\mathcal{N}$ a déjà été rencontré lors de la compilation (ligne 5). On utilise pour cela la fonction cache qui fait le lien entre les réseaux rencontrés et la racine du MDDG correspondant au réseau. Si $\mathcal{N}$ a déjà été trouvé durant la compilation, alors l'algorithme retourne simplement la racine du nœud de la formule MDDG correpondante.

Sinon on simplifie $\mathcal{N}$ en supprimant les contraintes universelles qu'elle contient (ceci est effectué par la fonction removeUniversal à la ligne 6). Puis, on cherche les composantes connexes du réseau obtenu (ligne 7). La fonction connectedComponents retourne une partition *CoCo* de l'ensemble courant des variables $\mathcal{X}$ correspondant aux composantes connexes du graphe primal de $\mathcal{N}$ (un simple parcours en largeur d'abord du graphe permet de les déterminer). Ensuite, on considère chaque élément *Co* de *CoCo* successivement (ligne 9). *Co* est donc un ensemble de variables indépendant des autres éléments de *CoCo* ; chaque réseau $\mathcal{N}$ correspondant peut donc être compilé séparément, que l'on regroupe dans un ensemble de nœuds $N_\wedge$ préalablement initialisé à l'ensemble vide (ligne 8). Pour chaque réseau *Co*, l'état de décision courant $\boldsymbol{s}$ peut être réduit aux variables apparaîssant dans *Co*. Une variable X_i est choisie grâce à la fonction selectVariable en ligne 10. Ensuite, en ligne 12, on considère successivement chacune des valeurs x_i du domaine courant de X_i. Plus précisément, pour chaque valeur x_i, on construit une affectation élémentaire correspondante $\{\langle X_i, x_i\rangle\}$. On conditionne le réseau par $\boldsymbol{s}$ que l'on a restreint aux variables de *Co* privées de X_i, auquel on

4. Dans notre implémentation, le conditionnement $\mathcal{N} \mid \boldsymbol{s}$ à la ligne 2 est fait implicitement (nous le présentons ainsi pour des raisons de clarté). A chaque étape, seules les listes des variables non instanciées et des domaines courants sont mises à jour (pour des raisons d'efficacité, les contraintes ne sont jamais modifiées).

ajoute l'affectation $\{\langle X_i, x_i\rangle\}$; ce réseau est compilé récursivement (ligne 13). L'ensemble $N_\vee$ des nœuds de décision, initialisé à l'ensemble vide (ligne 11), est mis à jour (ligne 13). Lorsque toutes les valeurs x_i du domaine ont été considérées, on construit un nouveau nœud de décision étiqueté par X_i que l'on ajoute à l'ensemble des nœuds $N_\wedge$ (ligne 14). Une fois traité l'ensemble des éléments de $CoCo$, on rasssemble les éléments de $N_\wedge$ en une conjonction sous forme d'un nœud $\wedge$ N grâce à la fonction aNode à la ligne 15. On associe ce nœud à l'entrée $\mathcal{N}$ du cache (ligne 16). Enfin, en ligne 17, nous retournons la racine du MDDG représentant $\mathcal{N}$.

Il est garanti que l'algorithme 1 termine, puisqu'à chaque étape de récursion, au moins une variable du CN initial est instanciée. Par construction, les solutions du MDDG en sortie sont exactement les mêmes que celles du CN en entrée.

5 Expérimentations

Alors que la compilation de CN est un sujet de recherche actif depuis des années (voir [27, 1, 12]), il n'existe pas à notre connaissance de compilateurs adaptés aux CNs de domaines non booléens et gérant des contraintes représentées en intension. En particulier, le compilateur AOMDD [5] suppose que toutes les contraintes du CN soient représentées en extension par leurs n-uplets autorisés. Heureusement, un grand nombre d'encodages CNF adapté aux CNs existent (entre autres [9, 3, 16, 25, 28, 14]), permettant de comparer notre approche à une compilation en Decision-DNNF précédée d'une transformation du réseau en CNF.

Notre cadre expérimental. Nous avons sélectionné 173 CNs issus de 15 familles différentes. Certaines instances contiennent des contraintes définies en extension, par leurs listes des n-uplets autorisés ou encore par leurs listes de n-uplets interdits. Pour d'autres instances, les contraintes sont données en intension.

Notre but fut de compiler chaque CN en une représentation en MDDG grâce au compilateur cn2mddg, et en une représentation en Decision-DNNF, en traduisant d'abord chaque instance en CNF, puis en utilisant le compilateur Dsharp, qui produit une représentation en Decision-DNNF à partir d'une formule CNF. Pour cela, nous avons considéré deux encodages CNF différents. Tout d'abord, le *sparse encoding*, qui consiste à encoder les domaines des variables, puis les contraintes suivant une méthode mixte (dans le but de minimiser le nombre de clauses générées, chaque contrainte est encodée en utilisant soit le *support encoding* soit le *conflict encoding*). Enfin, le *log encoding*, qui utilise un nombre logarithmique de variables booléennes pour encoder les domaines des variables et qui encode les contraintes suivant le *conflict encoding*. [6]

Pour chaque instance, nous avons calculé le temps de compilation (en secondes) et la taille de la formule compilée (le nombre d'arcs dans le graphe). Pour les approches basées sur une transformation en CNF, nous avons aussi calculé le temps de traduction et la taille de la formule CNF obtenue (le nombre de variables et de clauses). Nous avons lancé nos expérimentations sur un Quad-core Intel XEON X5550 avec 32Gio de mémoire. Pour chaque instance nous avons fixé un temps limite (*time-out*) à 3600 secondes pour la phase de transformation en CNF (ainsi que pour la phase de compilation) et un espace mémoire limite de 8Gio pour stocker la formule CNF (ainsi que pour la représentation de la forme compilée).

Quelques résultats obtenus. Dans le temps et la mémoire alloués, cn2mddg a réussi à compiler 131 instances sur les 173 testées. La compilation s'est terminée sur un *time-out* (TO) pour 32 instances, et sur un *memory-out* (MO) pour 10 instances. On peut observer un très grand écart de performances avec Dsharp qui n'a compilé que 83 instances quand le *sparse encoding* a été utilisé, et 61 instances lorsque le *log encoding* a été utilisé. Plus en détails, la transformation en CNF a conduit à 27 *memory-out* avec le *sparse encoding* (respectivement 35 avec le *log encoding*). Sur les 146 (respectivement 138) instances restantes, la compilation en Decision-DNNF par Dsharp s'est terminée sur un *time-out* pour 24 instances (respectivement 21), et sur un *memory-out* pour 39 instances (respectivement 56).

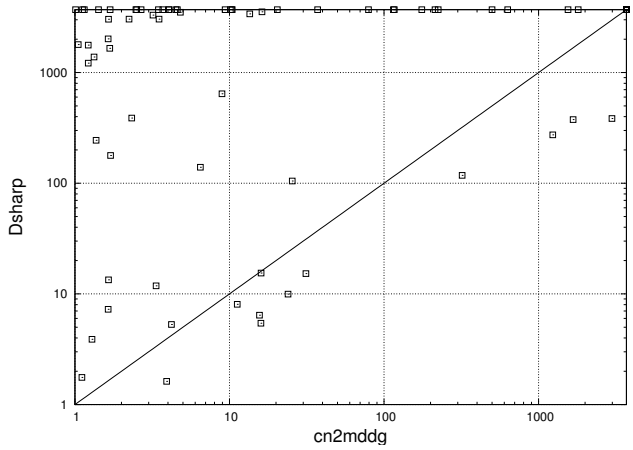
Quel que soit l'encodage considéré (*sparse encoding* ou *log encoding*) l'approche par transformation en CNF suivie d'une compilation en Decision-DNNF s'est révélée peu compétitive dans la plupart des cas. Ceci peut s'expliquer à la fois par le très grand nombre de variables booléennes apparaîssant dans la formule CNF générée, et de la perte d'information sur la structure du problème dû au format CNF (comparé à la représentation en CN). Notre compilateur cn2mddg s'est montré bien plus robuste puisqu'il a réussi à compiler beaucoup de CN qui se sont montrés hors de portée des approches par transformation en CNF puis compilation.

En particulier, chacune des 83 instances compilées avec succès par Dsharp (avec le *sparse encoding* en amont) se sont aussi montrées compilables en MDDG

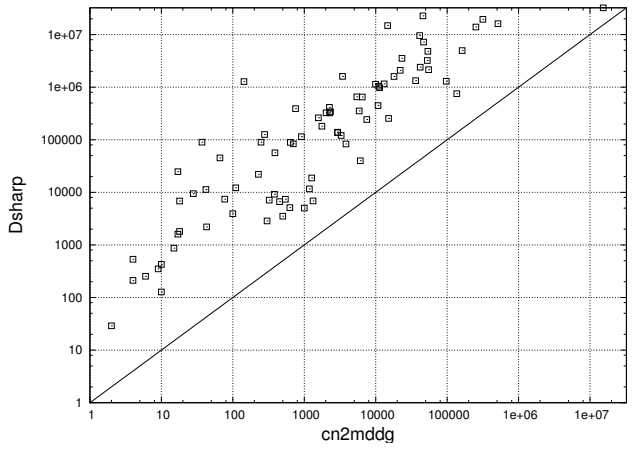
5. Disponible à l'adresse http://graphmod.ics.uci.edu/group/aomdd

6. Certains traducteurs disponibles, comme Sugar [24] ou Azucar [26], s'appuient sur des encodages qui ne préservent pas l'équivalence (ils sont orientés vers la résolution du problème de satisfaction), et ne peuvent donc être utilisés tels quels dans notre cadre.

en utilisant cn2mddg. Les temps de compilation requis pour créer les représentations en MDDG depuis les réseaux donnés en entrée sont souvent plus courts que les temps de compilation nécessaires pour créer les compilations en formules Decision-DNNF depuis les représentations CNF des réseaux donnés. Plus remarquablement encore, les tailles des formules compilées se révèlent toujours plus petites, parfois de plusieurs ordres de grandeur, quand le langage cible est MDDG comparé au langage Decision-DNNF.



(a) Temps



(b) Taille

FIGURE 3 – Comparaison entre temps/taille de compilation en MDDG et temps/taille de compilation en Decision-DNNF.

On peut observer cet écart clairement sur la figure 3, où chaque point représente l'une des 83 instances que Dsharp a réussi à compiler (avec le *sparse encoding* en amont). Le temps nécessaire pour compiler (respectivement la taille de) la représentation en MDDG est donnée sur l'axe des abscisses et le temps nécesssaire pour compiler (respectivement la taille de) la représentation en Decision-DNNF depuis la CNF encodée est donné sur l'axe des ordonnées. [7] Notez bien que toutes les échelles considérées sur le figure sont logarithmiques.

Le tableau 1 présente une sélection des résultats obtenus. Chaque ligne correspond à un réseau de contraintes CN $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ identifié par la colonne la plus à gauche. Les colonnes suivantes donnent respectivement le « type » de CN (en extension ou en intension), son nombre $\#\mathcal{X}$ de variables, son nombre $\#C$ de contraintes, l'arité maximale *maxA* des contraintes, la taille maximale *maxD* des domaines, un majorant *tw* de la largeur d'arbre de son graphe primal, [8] le temps nécessaire pour compiler le CN en MDDG en utilisant cn2mddg, et la taille de la formule compilée.

Pour les deux encodages en CNF que nous avons considérés, nous avons indiqué le nombre *#pv* de variables propositionnelles que contient la CNF, le nombre *#pcl* de clauses obtenues, le temps nécesssaire pour compiler la CNF en Decision-DNNF en utilisant Dsharp, et la taille de représentation en Decision-DNNF résultante.

Les résultats obtenus montrent que l'approche cn2mddg est plus compétitive que l'approche par transformation en CNF suivi d'une compilation en Decision-DNNF. La comparaison est, en effet, en faveur de cn2mddg que l'on considère comme critère de comparaison le nombre d'instances résolues, ou (ce qui est plus important) la taille des formules compilées. Les résultats montrent aussi que la compilation de CNs issus d'applications réelles, et possédant une complexité significative (souvent hors de portée de la compilation en *tree clustering* [10], vu la taille de leur domaine et la largeur d'arbre de leur graphe primal), vers le langage MDDG est réalisable.

Nous avons aussi réalisé une évaluation différentielle de chaque technique nouvelle utilisée dans cn2mddg afin de déterminer son apport. Pour des raisons d'espace, nous ne pouvons pas donner ici le détail de cette évaluation. Néanmoins, soit dom/wdeg+noU la version de cn2mddg pour laquelle l'heuristique de choix de variable *dom/wdeg* est utilisée (à la place de *bc*), et pour laquelle les contraintes universelles ne sont pas gérées spécifiquement ; notre évaluation a montré que dom/wdeg+noU a réussi à résoudre seulement 101 instances (sur 173) dans le temps et la mémoire alloués ; de plus, le nombre d'instances (sur 101) pour lesquelles la taille de la représentation en MDDG obtenue en utilisant cn2mddg (resp. dom/wdeg+noU) est inférieure à $p = \frac{1}{2}$ fois la taille de la représentation en MDDG obtenue en utilisant dom/wdeg+noU (resp. cn2mddg) est 35

7. Notons que la différence de temps en faveur de cn2mddg aurait été bien plus grande encore si nous avions pris en compte le temps nécessaire à l'encodage du réseau en CNF.

8. Calculé en utilisant QuickBB - voir http://www.hlt.utdallas.edu/~vgogate/quickbb.html - avec l'heuristique de choix de variable *random* et un *time-out* de 1800 secondes.

(resp. 6). Le nombre correspondant pour la proportion $p = \frac{1}{10}$ (au lieu de $\frac{1}{2}$) est 12 (resp. 0). Ainsi, pour plus de 10% des instances traitées, l'heuristique de choix de variable *bc* couplée à la gestion des contraintes universelles a conduit à une diminution de la taille de la forme compilée d'un ordre de grandeur.

6 Conclusion

Dans cet article, nous avons présenté un algorithme descendant `cn2mddg` pour la compilation de réseaux de contraintes à domaines finis, en graphes de décision décomposables multivalués. Parmi les différentes techniques qu'il incorpore, notre algorithme tire parti d'une méthode de *caching* spécifique, d'une nouvelle heuristique de choix de variable basée sur la *betweennes centrality*, et une gestion spécifique des contraintes universelles.

Notre large panel d'expérimentations a montré que `cn2mddg` permet de compiler des réseaux de contraintes qui ne peuvent être compilés en Decision-`DNNF` via un encodage préliminaire en `CNF`. Nous avons aussi montré que `cn2mddg` permet typiquement de produire des représentations compilées de bien plus petites tailles.

Ce travail ouvre de nombreuses perspectives. Parmi celles-ci, nous prévoyons d'étendre notre algorithme à la compilation de réseaux de contraintes pondérées, aux réseaux bayésiens et aux réseaux de Markov, et d'évaluer et de comparer le compilateur obtenu aux approches existantes pour traiter ces modèles graphiques.

Références

[1] J. Amilhastre, H. Fargier, and P. Marquis. Consistency restoration and explanations in dynamic CSPs application to configuration. *Artificial Intelligence*, 135(1-2) :199–234, 2002.

[2] J. Amilhastre, H. Fargier, A. Niveau, and C. Pralet. Compiling csps : A complexity map of (non-deterministic) multivalued decision diagrams. *International Journal on Artificial Intelligence Tools*, 23(4), 2014.

[3] O. Bailleux and Y. Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *Proc. of CP'03*, pages 108–122, 2003.

[4] F. Boussemart, F. Hemery, Ch. Lecoutre, and L. Saïs. Boosting systematic search by weighting constraints. In *Proc. of ECAI'04*, pages 146–150, 2004.

[5] U. Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30(2) :136–145, 2008.

[6] H. Cambazard, T. Hadzic, and B. O'Sullivan. Knowledge compilation for itemset mining. In *Proc. of ECAI'10*, pages 1109–1110, 2010.

[7] A. Darwiche. New advances in compiling cnf into decomposable negation normal form. In *Proc. of ECAI'04*, pages 328–332, 2004.

[8] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17 :229–264, 2002.

[9] J. de Kleer. A comparison of ATMS and CSP techniques. In *Proc. of IJCAI'89*, pages 290–296, 1989.

[10] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3) :353–366, 1989.

[11] H. Fargier and P. Marquis. On the use of partially ordered decision graphs in knowledge compilation and quantified Boolean formulae. In *Proc. of AAAI'06*, pages 42–47, 2006.

[12] H. Fargier and M.-C. Vilarem. Compiling CSPs into tree-driven automata for interactive solving. *Constraints*, 9 :263–287, 2004.

[13] E. Freuder and B. O'Sullivan. Grand challenges for constraint programming. *Constraints*, 19 :150–162, 2014.

[14] I. P. Gent. Arc consistency in SAT. In *Proc. of ECAI'02*, pages 121–125, 2002.

[15] J. Huang and A. Darwiche. The language of search. *Journal of Artificial Intelligence Research*, 29 :191–219, 2007.

[16] K. Iwama and S. Miyazaki. Sat-variable complexity of hard combinatorial problems. In *Proc. of IFIP World Computer Congress'94*, pages 253–258, 1994.

[17] U. Junker. Configuration. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 24. Elsevier, 2006.

[18] R. Mateescu and R. Dechter. Compiling constraint networks into AND/OR multi-valued decision diagrams (AOMDDs). In *Proc. of CP'06*, pages 329–343, 2006.

[19] R. Mateescu, R. Dechter, and R. Marinescu. AND/OR multi-valued decision diagrams (AOMDDs) for graphical models. *Journal of Artificial Intelligence Research*, 33 :465–519, 2008.

[20] Ch.J. Muise, Sh.A. McIlraith, J.Ch. Beck, and E.I. Hsu. Dsharp : Fast d-DNNF compilation

	CN							
Name	type	$\#\mathcal{X}$	$\#C$	maxA	maxD	tw	time	size
rect-packing/rect-packingrpp07-true	I	1168	1273	8	23	15	31.22	5875
rect-packing/rect-packingrpp08-true	I	1584	1714	9	31	17	1234.72	162258
rect-packing/rect-packingrpp09-true	I	2196	2353	10	36	19	1673.33	514754
ghoulomb/ghoulomb3-4-6	I	3524	3543	16	37	?	164.95	74427
ghoulomb/ghoulomb3-4-5	I	2033	2051	11	26	31	15.17	5162
driver/normalized-driverlogw-08c-sat-ext	E	408	9321	2	11	92	15.63	2931
driver/normalized-driverlogw-08cc-sat-ext	E	408	9321	2	11	92	15.96	2931
scheduling/talent-concert	I	325	352	46	316	52	1277.21	404437
fapp/fapp18/normalized-fapp18-0350-8	I	350	2387	2	302	187	0.69	4243
fapp/fapp19/normalized-fapp19-0350-6	I	350	3114	2	802	130	79.34	1694146
costaArray/CostasArray10	I	110	338	4	19	23	10.39	13440
costaArray/CostasArray14	I	210	808	4	27	36	TO	-
photo/photophoto2	I	89	133	21	11	21	499.93	9564220
photo/photophoto1	I	75	102	18	9	18	10.13	210646
rlfap/normalized-scen4	I	680	3967	2	44	30	3.47	52226
rlfap/normalized-scen7-w1-f4	I	400	660	2	40	7	4.60	444483
radiation/radiation04	I	781	569	9	5180	33	-	MO
renault/normalized-renault-mod-32-ext	E	111	154	10	42	11	20.39	160238
renault/normalized-renault-mod-11-ext	E	111	149	10	42	10	16.22	41919
still-life/still-life7x7	I	690	803	50	50	49	1819.87	738478
configit/Aralia/edfpa15r	I	198	110	13	2	28	175.49	2044261
configit/Aralia/edfpa14q	I	505	194	22	2	34	TO	-
configit/Aralia/das9207	I	600	324	8	2	15	8.94	45853

	CNF - sparse mixed encoding				CNF - log conflict encoding			
Name	#pv	#pcl	time	size	#pv	#pcl	time	size
rect-packing/rect-packingrpp07-true	8709	110661	15.26	353048	2378	70087	37.70	976110
rect-packing/rect-packingrpp08-true	17724	248417	273.73	4973120	3225	147307	1139.74	7938679
rect-packing/rect-packingrpp09-true	37044	593518	375.66	16118647	4466	392657	TO	-
ghoulomb/ghoulomb3-4-6	MO	MO	MO	MO	MO	MO	MO	MO
ghoulomb/ghoulomb3-4-5	MO	MO	MO	MO	MO	MO	MO	MO
driver/normalized-driverlogw-08c-sat-ext	9528	62825	6.42	139306	1050	46081	32.78	499796
driver/normalized-driverlogw-08cc-sat-ext	9528	62825	5.43	136501	1050	46081	23.63	425617
scheduling/talent-concert	MO	MO	MO	MO	MO	MO	MO	MO
fapp/fapp18/normalized-fapp18-0350-8	9199429	63582486	-	MO	2810	52179077	-	MO
fapp/fapp19/normalized-fapp19-0350-6	166130802	867243022	-	MO	MO	MO	MO	MO
costaArray/CostasArray10	149564	841930	TO	-	540	3606946	TO	-
costaArray/CostasArray14	988671	5568047	TO	-	1036	34687218	-	MO
photo/photophoto2	685555	14326576	TO	-	204	10923133	-	MO
photo/photophoto1	73095	1328839	TO	-	172	1117281	TO	-
rlfap/normalized-scen4	915553	4875002	-	MO	4060	3058032	TO	-
rlfap/normalized-scen7-w1-f4	102556	683178	-	MO	2392	522995	-	MO
radiation/radiation04	MO	MO	MO	MO	MO	MO	MO	MO
renault/normalized-renault-mod-32-ext	222582	1755876	TO	-	286	138124077	-	MO
renault/normalized-renault-mod-11-ext	223718	1762294	3538.01	2399273	286	138117804	-	MO
still-life/still-life7x7	MO	MO	MO	MO	MO	MO	MO	MO
configit/Aralia/edfpa15r	396	24710	-	MO	396	24710	-	MO
configit/Aralia/edfpa14q	1010	5793030	TO	-	1010	5793030	TO	-
configit/Aralia/das9207	1200	3894	642.68	22707412	1200	3894	97.86	9937506

TABLE 1 – Une sélection de résultats expérimentaux.

with sharpSAT. In *Proc. of AI'12*, pages 356–361, 2012.

[21] U. Oztok and A. Darwiche. On compiling CNF into decision-DNNF. In *Proc. of CP'14*, pages 42–57, 2014.

[22] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier, 2006.

[23] O. Roussel and Ch. Lecoutre. XML Representation of Constraint Networks : Format XCSP 2.1. Technical report, Computing Research Repository (CoRR) abs/0902.2362, feb 2009.

[24] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2) :254–272, 2009.

[25] T. Tanjo, N. Tamura, and M. Banbara. A compact and efficient sat-encoding of finite domain CSP. In *Proc. of SAT'11*, pages 375–376, 2011.

[26] T. Tanjo, N. Tamura, and M. Banbara. Azucar : A sat-based CSP solver using compact order encoding - (tool presentation). In *Proc. of SAT'12*, pages 456–462, 2012.

[27] N.R. Vempaty. Solving constraint satisfaction problems using finite state automata. In *Proc. of AAAI'92*, pages 453–458, 1992.

[28] T. Walsh. SAT v CSP. In *Proc. of CP'00*, pages 441–456, 2000.

Résolution de SCSP avec borne de confiance pour les jeux de stratégie

Frédéric Koriche, Sylvain Lagrue, Éric Piette, Sébastien Tabary

Université Lille-Nord de France CRIL - CNRS UMR 8188 Artois, F-62307 Lens
{koriche,lagrue,epiette,tabary}@cril.fr

Résumé

Cet article examine un fragment du problème de satisfaction de contraintes stochastiques, représentant les jeux à informations complètes et incertaines. Nous proposons un algorithme de résolution permettant de trouver des stratégies gagnantes pour cette classe de jeux. L'intérêt de ce travail est de permettre la résolution efficace d'un SCSP initial par la résolution successive d'une séquence de « micro-SCSP » à chaque étape du jeu. L'algorithme MAC, proposé pour résoudre les premiers micro-SCSP de la séquence, est couplé à une heuristique de type UCB utilisée pour estimer les valeurs des stratégies sur toute la séquence. Notre approche, MAC-UCB, est validée par une série d'expérimentations sur un ensemble de jeux très divers, décrits en GDL (Game Description Language), et comparée à l'algorithme UCT, qui est la référence actuelle pour cette classe de jeux.

1 Introduction

L'objectif du *General Game Playing* (GGP) est de concevoir des algorithmes de décision, non pas « dédiés » à un jeu de stratégie particulier, mais « génériques » de par leur capacité à jouer de manière pertinente à une grande variété de jeux. Une compétition est d'ailleurs organisée chaque année par AAAI [7], au cours de laquelle de nombreux algorithmes de jeux s'y confrontent. Les jeux sont décrits dans un langage de représentation, appelé GDL pour *Game Description Language* [10] ; la dernière version de ce langage, GDLII, permet de modéliser des jeux à information incertaine et incomplète [20]. Parmi les algorithmes génériques de jeu, on compte notamment des méthodes de type Monte Carlo [6], la construction automatique de fonctions d'évaluation [4], ou encore la programmation logique [19] et ASP [13]. Bien entendu, le problème GGP n'est pas cantonné aux « univers ludiques » : il permet de modéliser des problèmes de prise de décision séquentielle mono ou multi-agents.

Par son approche déclarative de la résolution de problèmes combinatoires, la programmation par contraintes offre un cadre prometteur pour relever le défi du GGP. Parmi les différents formalismes à base de contraintes proposés pour représenter et résoudre des jeux, on peut citer les QCSP [8], les *strategic* CSP [3] ou encore les *constraint games* [15]. Cependant, la plupart de ces formalismes sont restreints aux jeux à information complète et certaine : à chaque instant, les joueurs ont une information complète de l'état du jeu et leurs actions sont déterministes. Cette étude se focalise sur le formalisme des réseaux de contraintes stochastiques (SCSP) [21], utilisés pour la prise de décision séquentielle, dont l'effet incertain des actions est modélisé par une distribution de probabilités.

Nous examinons un fragment de SCSP, proposé dans [9], permettant de modéliser des jeux GDL à information complète mais incertaine. De manière importante, le SCSP associé à cette classe de jeux peut être décomposé

en une séquence de « micro-SCSP ». A partir de cette décomposition, nous proposons un algorithme de décision séquentielle couplant la méthode MAC (*Maintaining Arc Consistency*) pour résoudre les micro-SCSP, et la méthode des bandits stochastiques UCB (*Upper Confidence Bound*) pour estimer l'utilité des stratégies. Notre algorithme, MAC−UCB, évalué sur un grand nombre d'expérimentations ($\sim 10^5$) réalisées sur des jeux GDL très divers, se révèle être, dans la plupart des scénarios, meilleur que l'algorithme UCT (*Upper Confidence Tree*), qui demeure la référence dans le domaine des jeux de stratégie à information incertaine.

L'article est organisé de la manière suivante. Le cadre SCSP pour les jeux GDL est introduit en Section 2, et l'algorithme MAC−UCB est détaillé en Section 3. Avant de conclure, la Section 4 présente sur une série de jeux GDL avec des temps de réflexion différents, les résultats obtenus par MAC−UCB représentant un joueur, face à UCT représentant le joueur adverse.

2 Un fragment de SCSP pour GDL

Les SCSP examinés dans cette étude étendent le modèle originel de Walsh ([21]) pour traiter les contraintes valuées.

SCSP. De manière formelle, un *Stochastic Constraint Satisfaction Problem* (SCSP) est un tuple $\langle X, Y, D, P, \mathcal{C}, \theta \rangle$ tel que X est un ensemble ordonné de n variables et Y est le sous-ensemble de X spécifiant les variables stochastiques ; D représente l'ensemble des domaines associés aux variables de X, P est l'ensemble des distributions de probabilité appliquées aux domaines des variables stochastiques, $\mathcal{C}$ est l'ensemble des contraintes, et θ est la valeur de seuil comprise entre 0 et 1.

Les variables incluses dans $X \setminus Y$ sont appelées variables de décision et ont la même signification que celles définies dans un CSP classique. Nous notons $D(z)$ le domaine associé à une variable $z \in X$. Plus généralement, pour un sous-ensemble $Z = \{z_1, \cdots, z_k\} \subseteq X$, nous notons $D(Z)$ l'ensemble des tuples de valeurs $D(z_1) \times \cdots \times D(z_k)$.

Chaque contrainte d'un SCSP est une paire $c = \langle scp_c, val_c \rangle$, où scp_c est le sous-ensemble de X, dénommé *portée* de c, et val_c est une fonction qui associe pour chaque tuple $\boldsymbol{\tau} \in D(scp_c)$ une valeur (ou utilité) $val_c(\boldsymbol{\tau}) \in [0, 1] \cup \{-\infty\}$. Une contrainte de décision est une contrainte $c \in C$ où scp_c est restreint aux variables de décisions. À la différence d'une contrainte stochastique dont scp_c porte au moins sur une variable stochastique. Une contrainte $c \in C$ est *dure* si val_c retourne $-\infty$ si c est violée sinon 0 et *souple* si la valeur de retour de la fonction val_c est comprise entre 0 et 1. Si c est une contrainte dure, alors elle peut être représentée de manière usuelle par une relation, notée rel_c, qui contient l'ensemble des tuples autorisés pour scp_c (i.e. les tuples $\boldsymbol{\tau}$ pour lesquels $val_c(\boldsymbol{\tau}) \neq -\infty$).

Étant donné un ensemble de variables $Z = \{z_1, \ldots, z_k\} \subseteq X$, une *instanciation* I de Z est un ensemble de paires variable-valeur $\{(z_1, v_1), \ldots, (z_k, v_k)\}$, tel que $v_i \in D(z_i)$ pour chaque $z_i \in Z$. Une instanciation est dite *complète* si $Z = X$. Pour un sous-ensemble Z' de Z, nous notons $I_{|Z'}$ la projection de I sur Z'. L'utilité de I est donnée par :

$$val(I) = \sum_{c \in \mathcal{C}} val_c(I_{|scp_c})$$

Une politique π est un ensemble d'instanciations complètes, structurées en arbre, dont les nœuds sont les variables de X, et dont les arêtes sont étiquetées par la valeur assignée au nœud parent. Les nœuds associés aux variables de décision ont un seul fils, tandis que les nœuds associés aux variables stochastiques ont un fils pour chaque valeur possible de leurs domaines. Ainsi, chaque instanciation complète de π est identifiée par un chemin depuis la racine de π jusqu'à une feuille. Les feuilles sont étiquetées par l'utilité du chemin correspondant. L'utilité espérée d'une politique π est définie par la somme des utilités des feuilles pondérées par leurs probabilités.

Une *solution* du SCSP est une politique π dont l'utilité espérée est plus grande ou égale au seuil θ. Comme $\theta > -\infty$, les contraintes dures sont nécessairement satisfaites par une solution. Ainsi, une politique π satisfait le SCSP si toutes les contraintes dures sont satisfaites et si l'utilité espérée excède le seuil θ.

Exemple 1 *Considérons le SCSP défini par*

les variables de décision x_1 et x_2 et la variable stochastique y ; les trois variables sont définies sur le domaine $\{0, 1, 2\}$, et la distribution de y est uniforme ($P(0) = P(1) = P(2) = 1/3$). Le réseau possède deux contraintes :

$$c_h(x_1, x_2) = \begin{cases} 0 & si\ x_1 = x_2 \\ -\infty & sinon \end{cases}$$

$$c_s(x_1, y) = \begin{cases} 1 & si\ x_1 + y \geq 1 \\ 0 & sinon \end{cases}$$

Enfin, le seuil θ est fixé à $1/2$. La politique π représentée par la figure 1 est une solution du SCSP, puisqu'elle ne viole pas la contrainte dure c_h et satisfait la contrainte valuée c_s avec une utilité espérée de $2/3 \geq \theta$.

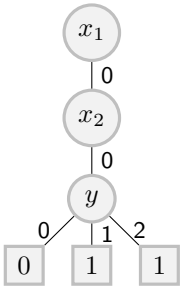


FIGURE 1 – Une politique π

Le problème consistant à trouver une politique solution d'un SCSP est, en général, PSPACE-complet. Il existe cependant une sous-classe intéressante de SCSP pour laquelle le problème de satisfaction est de complexité moindre ($\mathsf{NP}^{\mathsf{PP}}$). Il s'agit des *one stage* SCSP [21], ou micro-SCSP, que l'on note ici μSCSP. Formellement, un μSCSP est un SCSP $(X, Y, D, P, \mathcal{C}, \theta)$, dans lequel la restriction suivante est imposée sur les politiques : un arbre d'instanciations complètes π est une politique de μSCSP si toutes les variables de décision X sont assignées *avant* les variables stochastiques Y. Comme le montre Walsh, tout SCSP défini sur n variables de décision peut être vu comme une séquence de n μSCSP (les contraintes peuvent néanmoins porter sur plusieurs μSCSP).

SCSP d'un programme GDL. Dans [9], un formalisme a été proposé pour traduire des jeux GDL [10] vers un SCSP. Un programme GDL est un ensemble de règles en logique du premier ordre ; pour un horizon T donné, ce programme peut être instancié en T ensembles de règles, définis aux tours $t \in \{1, \cdots, T\}$.

Nous présentons ici de manière informelle le processus de transformation d'un programme GDL à horizon T en SCSP. Pour chaque tour de jeu t, l'ensemble des règles instanciées à t est représenté par un μSCSP_t défini sur l'ensemble des variables $X_t \cup X_{t+1} \cup \{y_t\}$. Deux variables de décision $\mathsf{terminal}_t$ et $\mathsf{control}_t$ sont utilisées pour capturer l'état terminal du jeu et indiquer quel joueur à la main. Chaque fluent apparaissant à l'instant t est associé à deux variables de décision, correspondant aux états du fluent à l'instant t et à l'instant $t + 1$. De la même manière, chaque action apparaissant à t est associée à une variable de décision. Les domaines de ces variables représentent toutes les combinaisons possibles de constantes qui sont instanciées par les atomes du programme GDL. Enfin, chaque règle est associée à une contrainte dont la portée est extraite de façon similaire aux domaines des variables pour identifier toutes les combinaisons de constantes autorisées. En substance, les contraintes représentent les actions autorisées à l'instant t et les transitions possibles entre t et $t + 1$.

Le comportement incertain de l'environnement (appelé random dans GDL) est capturé par la variable stochastique y_t (ex : résultat d'un jet de dés à l'instant t). Le domaine de y_t est donné par les actions possibles de l'environnement à l'instant t et, selon les spécifications GDLII [20], la distribution de probabilités $P(y_t)$ est définie comme uniforme sur son domaine de valeurs [1].

Par construction, le SCSP associé à un programme GDL à horizon T est défini comme une séquence de micro-SCSP $\langle \mu\text{SCSP}_1, \cdots, \mu\text{SCSP}_T \rangle$, dans laquelle les contraintes portant sur les coups légaux et les transitions d'états sont spécifiées pour chaque μSCSP_t, $t \in \{1, \cdots, T-1\}$. Le dernier

1. Néanmoins, et comme précisé dans [20], il est possible de représenter des distributions non-uniformes d'actions possibles en autorisant plusieurs valeurs associées à un choix d'action dans le domaine de y_t.

micro-SCSP de la séquence (μSCSP$_T$), défini sur les variables $X_T \cup \{y_T\}$, joue un rôle particulier : dans un jeu GDL, le score n'est donné qu'à l'horizon T et donc seul μSCSP$_T$ possède des contraintes valuées capturant la fonction de score (dans $[0, 1]$) sur l'état terminal. Cette structuration particulière du SCSP incite naturellement à résoudre le problème de manière séquentielle en explorant itérativement chaque μSCSP$_t$.

Le seuil θ peut être ajusté selon la stratégie désirée : en partant d'un seuil de 0 qui autorise toute politique « faisable » (ne violant aucune contrainte dure), il est possible de construire des SCSP de plus en plus contraints, dont la résolution pour un seuil θ peut tirer parti de la résolution pour un seuil $\theta' < \theta$. En particulier, les solutions des μSCSP$_t$ aboutissant à des stratégies non-optimales pour θ' peuvent être élaguées lors de la résolution du problème au seuil θ.

3 MAC−UCB

Dans cette section, nous présentons notre algorithme de résolution MAC−UCB. A partir d'un SCSP décomposé en une séquence $\langle \mu\text{SCSP}_t \rangle$ de micro-problèmes de résolution de contraintes stochastiques, MAC−UCB cherche une politique solution en maintenant l'arc consistance sur chaque μSCSP$_t$ et en échantillonnant avec borne de confiance l'utilité des solutions du μSCSP$_t$.

Prétraitement. Avant de résoudre un μSCSP, nous utilisons des techniques de prétraitement pour améliorer l'efficacité de résolution. Nous commençons par fusionner les contraintes dures de même portée. Étant donné un μSCSP P, deux contraintes dures c_i et c_j de P telle que $scp_{c_i} = scp_{c_j}$ deviennent une unique contrainte c_k où $rel_{c_k} = rel_{c_i} \cap rel_{c_j}$ et $scp_{c_k} = scp_{c_j} = scp_{c_i}$. Nous supprimons aussi toutes les contraintes unaires (e.g. contraintes c dont $|scp_c| = 1$). Le domaine de la variable impliquée par la contrainte unaire est restreint aux valeurs autorisées par les tuples de la relation associée.

Ensuite, nous identifions les variables universelles dans chaque contrainte. Une variable est universelle sur c si quelque soit la valeur assignée, c est toujours satisfaite. Autrement dit, une variable $x \in scp_c$ est universelle si $|rel_c|$ est égal au produit de la taille du domaine de x avec le nombre de tuples de la relation associée à la contrainte c_i, telle que $scp_{c_i} = scp_c \setminus \{x\}$. Ces variables sont supprimées de la portée des contraintes.

La dernière technique de prétraitement est d'établir la propriété de singleton arc consistance (*Singleton Arc Consistency* noté SAC [5]) sur le réseau de contraintes associé au μSCSP correspondant. Un réseau de contraintes P est singleton arc-consistant si chaque valeur de P est singleton arc-consistante. Une valeur est singleton arc-consistante si une fois assignée à sa variable, elle ne produit pas un réseau arc inconsistant. Nous appliquons cette propriété pour supprimer les valeurs inconsistantes des domaines des variables.

MAC. Une fois le micro-SCSP prétraité, le problème principal est d'énumérer les solutions faisables de ce problème, dont certaines peuvent aboutir à une solution optimale. A notre connaissance, la meilleure méthode pour résoudre un μSCSP est *Forward Checking* (FC) présenté dans [2]. Malheureusement pour un μSCSP contenant un grand nombre de contraintes, FC n'est pas suffisament efficace, à cause de sa faible capacité de filtrage. L'idée de notre approche est de résoudre le μSCSP avec l'algorithme de maintien d'arc consistance (MAC) [16, 17]. Pour rappel, l'algorithme MAC alterne inférence et recherche. Il effectue une recherche en profondeur d'abord avec retour arrière et maintient à chaque nœud une consistance locale : *la consistance d'arc* (AC) [12, 11]. Pour appliquer cet algorithme sur un problème de contraintes stochastiques, nous avons besoin de partitionner les contraintes : d'une part les contraintes de décision (CSP P') et d'autre part les contraintes stochastiques (μSCSP P''). L'algorithme MAC est appliqué sur un CSP classique composé uniquement des contraintes de P'. Les solutions du μSCSP sont identifiées en combinant la résolution du CSP P' avec celle du μSCSP P''.

Premièrement, nous commençons par résoudre le plus petit problème, le μSCSP P''. En effet, pour des jeux GDL, P'' contient seulement

une contrainte sur l'action de l'environnement et une contrainte pour chaque modification de fluent connecté aux actions de l'environnement. Cette partie du problème peut être traitée par une variante de *Forward Checking* (FC) [2] adaptée à notre problème. Comme notre algorithme est utilisé pour la décision séquentielle, notre version énumère toutes les politiques faisables de P'' alors que l'algorithme d'origine ne retourne que la valeur de satisfaction obtenue. L'ensemble des politiques solutions obtenues avec FC sur P'' est exprimable par une contrainte c_s dont la portée est l'ensemble des variables de décisions et la relation possède les valeurs des stratégies gagnantes. Pour réaliser la jointure des deux ensembles P' et P'' (projeté sur les variables de P') nous ajoutons à P' la contrainte c_s. De cette manière, quand nous utilisons l'algorithme classique MAC sur P', il retourne l'ensemble des solutions de P' qui seront les mêmes que celle du μSCSP d'origine. Cette partie de l'arbre de recherche est filtrée par la propriété d'arc consistance pour énumérer rapidement les solutions de P'.

Exemple 2 *Afin d'illustrer la résolution d'un μSCSP P, considérons le μSCSP d'origine donné par les variables de décision $\{x, z\}$ et la variable stochastique y. Le domaine de x est $\{1, 2, 3\}$, et le domaine de y et z est $\{0, 1, 2\}$. La distribution P de y est uniforme sur son domaine. Le réseau possède trois contraintes données par :*

$$c_1(x, y) : x + y > 1$$
$$c_2(y, z) : y + z > 1$$
$$c_3(x, z) : x = z$$

Enfin le seuil θ est fixé à $7/10$.

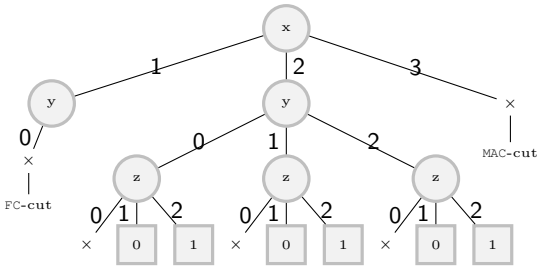
Après partitionnement, la partie CSP P' est donnée par les deux variables de décision x et z et la contrainte c_3. La partie μSCSP P'' est donnée par les trois variables x, y, z et les contraintes c_1 et c_2.

L'algorithme FC retourne pour le problème stochastique les politiques $(x = 2\,;\, z = 2)$ et $(x = 3\,;\, z = 2)$. La satisfaction associée pour ces politiques est de 1. Ainsi la contrainte c_s est ajoutée à P' est :

$$c_s(x, z) : x = z = 2 \mid x = z = 3$$

L'algorithme MAC retourne pour ce problème la solution : $(x = 2\,;\, z = 2)$.

La figure ci-dessous illustre l'arbre de recherche obtenu quand nous résolvons ces deux problèmes.



UCB. La résolution des différents μSCSP correspondant aux états d'un jeu GDL n'est pas suffisante. Comme nous l'avons vu précédemment dans la section 2, seul le dernier μSCSP de la séquence (μSCSP_T) possède des contraintes valuées et retourne une utilité comprise dans $[0, 1]$. C'est pourquoi après chaque résolution de μSCSP_t, nous avons besoin de simuler les états suivants afin d'estimer l'utilité des actions accomplies jusqu'à t. Dans cet objectif, nous utilisons l'algorithme standard des bandits stochastiques *Upper Confidence Bound* (UCB) [1] qui échantillonne les états suivants de $t+1$ jusqu'à T, et retourne une borne de confiance sur l'utilité des actions accomplies jusqu'à t. Rappelons qu'UCB commence par jouer chaque action possible une fois. Puis, il choisit une action i qu'il maximise par $\bar{x}_i + \sqrt{(2 \ln n)/n_i}$, où $\bar{x}_i$ est l'espérance empirique des utilités obtenues à partir de l'action i, n_i est le nombre de fois où l'action i a été choisie jusqu'ici, et n est le nombre total de simulations. Pour nos problèmes, $10,000$ simulations sont réalisées et UCB calcule une utilité avec ces données. Finalement, nous continuons la résolution du problème en instanciant le prochain μSCSP avec les valeurs obtenues sur la politique possédant l'utilité la plus grande, afin d'explorer l'arbre de recherche optimalement.

Élagage. Rappelons que la tâche de décision séquentielle associée à un jeu de stratégie est un problème d'optimisation. De manière classique, ce problème peut être géré par une séquence de

problèmes de satisfaction (SCSP) dont le seuil est augmenté progressivement.

Le seuillage progressif du problème d'optimisation est exploité dans MAC−UCB par deux méthodes d'élagage. La première est probabiliste et tire parti de la borne de confiance d'UCB ; rappelons que la qualité de chaque solution du problème P associé à un μSCSP est estimée par échantillonnage. Lorsqu'un nombre suffisant de valeurs de la variable stochastique se situent en dessous de la borne de confiance, l'instanciation des variables de décision, couplée avec les valeurs de la variable stochastique, est supprimée des solutions du sous-problème stochastique P'' de P. Par répercussion, le nombre de tuples de la contrainte c_s associée aux solutions de P'' projetées sur les variables de décision de P est réduit, ce qui facilite la résolution par MAC du problème déterministe P'. En d'autres termes, plus le seuil θ augmente, plus rapide est la résolution du μSCSP.

La deuxième forme d'élagage est déterministe, et consiste simplement à ajouter à UCB un cache lui permettant de connaître les feuilles déjà explorées [2]. En combinant cette mise en cache avec la connaissance du nombre de coups légaux à chaque état, il est possible de déduire si tout le sous-arbre après cet état a déjà été exploré ou non. Ainsi lorsqu'une solution d'un μSCSP donne lieu à un sous-arbre dont la valeur espérée est en dessous de θ, cette solution peut être éliminée avec certitude. En augmentant itérativement le seuil θ, UCB peut exploiter son cache pour éliminer dans chaque μSCSP les solutions sous-optimales pour θ.

4 Résultats expérimentaux

Après un tour d'horizon des SCSP et de MAC−UCB, nous sommes à présent en mesure de présenter les résultats expérimentaux obtenus sur divers jeux avec différents délais de décision pour les joueurs MAC−UCB et UCT.

Benchmarks. Afin de répondre à l'objectif du *General Game Playing* (GGP) [14], nous avons choisi 13 jeux très divers, décrits en GDLII, qui varient selon le type d'environnement (déterministe vs. stochastique), le nombre d'actions et de fluents, le nombre et la taille des contraintes induites par les règles et, naturellement, la fonction objectif (contrainte souple) associée au but.

— *Awale* est un jeu de stratégie à 2 joueurs avec 48 graines et 2 rangées de 6 trous. Chaque joueur contrôle les 6 trous de son coté du plateau. Le jeu commence avec 4 graines dans chaque trou. Le but du jeu est de capturer plus de graines que son opposant. A chaque tour, un joueur choisit un des 6 trous sous son contrôle et distribue toutes les graines présentes dans les trous suivants.

— *Backgammon* est un célèbre jeu de plateau à 2 joueurs avec 2 dés et 15 pièces par joueur. Les pièces sont déplacées selon un lancé de dés et un joueur gagne quand il supprime toutes les pièces de son plateau.

— *Bomberman* est un jeu de labyrinthe très populaire dans l'univers vidéo-ludique. Le but est de placer des bombes pour tuer tous les ennemis et détruire les obstacles.

— *Can't stop* est un jeu de plateau avec 4 dés et 3 moines par joueur, notre version du jeu impliquant seulement 2 joueurs. Le plateau inclut 9 montées de différentes tailles, le but étant d'atteindre le haut des 3 montées choisies par les moines.

— *Checkers* est le célèbre jeu de dames, implémenté ici sur un damier 8×8, avec 2 joueurs contrôlant leurs pions, le but étant d'éliminer ou bloquer les pions de l'adversaire.

— *Chess* est l'incontournable jeu d'échecs sur échiquier de 64 cases, avec 2 joueurs contrôlant 16 pièces, le but étant de mettre « échec et mat » le roi adverse, situation de prise sans qu'aucune action puisse y remédier.

— *Kaseklau* est un petit jeu de plateau à 2 joueurs avec un chat et une souris. Le but est de lancer les 2 dés pour avancer le chat et la souris sur différentes cases en ramassant les fromages placés par dessus quand le chat n'attrape pas la souris.

— *Orchard* ou « verger » est un jeu coopératif de plateau à 2 joueurs utilisant un dé à 6 faces. A chaque tour, si le dé tombe sur un des 4 arbres, le joueur qui a la main enlève un

2. Dans nos expérimentations, 32 GB ont été alloués pour le cache et cette limite n'a jamais été atteint.

fruit de l'arbre, si le dé tombe sur « panier », le joueur prend deux fruits de son choix, et si le dé tombe sur « corbeau », alors ce dernier perd une pièce. Le but du jeu est de récupérer collectivement tous les fruits avant que le corbeau perde toutes ses pièces.

— *Othello* est encore un jeu populaire de plateau 8×8 à 2 joueurs disposant de 64 pions bicolores. Le jeu s'arrête lorsqu'un aucun joueur ne peut plus placer de pion sur le plateau, le gagnant étant celui qui a placé le plus de pions.

— *Pacman* est un célèbre jeu d'arcade où le joueur contrôle *Pac-Man* au travers d'un labyrinthe mangeant des points et des fruits et évitant 4 fantômes souhaitant attraper le personnage. Le joueur gagne la partie s'il mange tous les points sans se faire attraper.

— *Pickomino* est un jeu de 8 dés et de 16 tuiles incluant 1 à 4 vers. A chaque tour, le joueur obtient un score issu du lancé de dés correspondant à la valeur d'une tuile. Le but est d'obtenir le maximum de vers en récupérant les tuiles contenant le plus de vers.

— *Tic-tac-toe* est un jeu déterministe bien connu avec 2 joueurs (X et O) qui marquent une grille de taille 3×3.

— *Yahtzee* est un jeu à 2 joueurs où le but est d'obtenir le plus grand score en lançant 5 dés. À chaque tour, les dés sont lancés jusqu'à 3 fois afin d'obtenir l'une des 13 combinaisons, chacune étant associée à un score.

Le tableau 1 présente les paramètres du μSCSP correspondant à la traduction de ces jeux et le temps nécessaire pour l'obtenir. Nous indiquons le nombre de variables, la taille maximum des domaines, le nombre de contraintes et le temps de traduction en secondes. Le jeu le plus difficile à traduire est *Backgammon* possédant un grand nombre de variables, chacune avec un domaine important et un nombre important de contraintes de grande portée. *Awale*, *Can't Stop*, *Chess*, *Pickomino* et le *Yahtzee* sont aussi intéressants de part la taille du domaine ou le nombre de contraintes.

Protocole. Dans le but de confronter notre algorithme à UCT (*Upper Confidence Tree*), nous avons implémenté un version de ce dernier, fondée sur l'analyse des jeux multi-joueurs [18]. Par souci d'équité, nous avons aussi ajouté un cache à UCT, lui permettant tout comme MAC−UCB de connaître par avance les sous-arbres qui ont déjà été explorés. Nous avons réalisé 390,000 instances de « matchs » entre UCT et MAC−UCB sur l'ensemble des jeux. Pour chaque jeu, un joueur suit la stratégie d'UCT et l'autre joueur celle de MAC−UCB. 5,000 instances de jeux sont réalisées sur différents temps (10s, 20s, 30s, 40s, 50s, 60s) par action.

Jeu	#vars	maxDom	#const	temps
Awale	19	37	73	94
Backgammon	76	768	86	347
Bomberman	145	64	42	31
Can't Stop	16	1,296	409	248
Checkers	86	262,144	52	43
Chess	71	4,096	50	76
Kaseklau	18	7,776	106	35
Orchard	9	146,410	40	12
Othello	81	65	29	51
Pacman	93	64	22	36
Pickomino	29	1,679,616	223	172
TicTacToe	19	18	37	0
Yahtzee	12	30	8,862	182

TABLE 1 – les jeux traduits en μSCSP et leurs caractéristiques.

Résultats. Le tableau 2 présente le pourcentage de victoires pour ces matchs sur chaque jeu avec 30 secondes par action. Nous indiquons aussi l'écart type (σ) correspondant au pourcentage de victoires de MAC−UCB. Pour tous les jeux, il apparaît que MAC−UCB est en moyenne plus performant qu'UCT, le phénomène s'accentuant en fait pour des délais plus grands. Les résultats sont particulièrement marquants sur les jeux stochastiques, tels que *Bomberman*, *Can't Stop*, *Kaseklau*, *Pacman*, *Pickomino*, *Yahtzee*, et surtout le *Backgammon*, pour lequel MAC−UCB gagne dans environ 70 % des cas. Pour jeux déterministes, l'écart est moins important (excepté *Othello*), ce qui s'explique par le fait que, sans action stochastique, MAC−UCB ne peut pas bénéficier du filtrage probabiliste. Enfin, pour le jeu coopératif *Orchard*, les deux algorithmes atteignent une moyenne de 70 %, Il est connu que ce pourcentage est le maximum possible pour la stratégie optimale.

Jeu	UCT	MAC-UCB	σ
Awale	43.3 %	56.7 %	1.63 %
Backgammon	31.8 %	68.2 %	5.49 %
Bomberman	34.6 %	65.4 %	6.32 %
Can't Stop	28.3 %	71.7 %	6.43 %
Checkers	38.8 %	61.2 %	2.12 %
Chess	46.2 %	53.8 %	1.75 %
Kaseklau	28.6 %	71.4 %	6.56 %
Orchard [3]	70.2 %	70.2 %	3.41 %
Othello	20.7 %	79.3 %	1.41 %
Pacman	30.9 %	69.1 %	2.73 %
Pickomino	36.6 %	63.4 %	4.89 %
Tictactoe	34.3 %	65.7 %	0.89 %
Yahtzee	28.8 %	71.2 %	5.48 %

TABLE 2 – Résultats pour les différents jeux GDL avec 30 secondes par coup

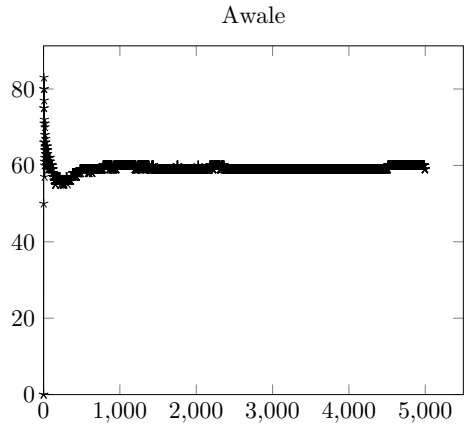


FIGURE 2 – En abscisse, le nombre d'instance de jeux et en ordonnée, le pourcentage de victoires de MAC-UCB contre UCT avec 30 secondes par coup sur *Awale*.

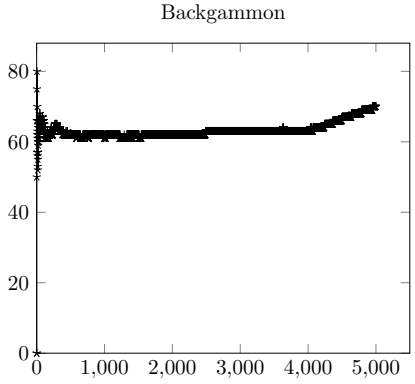


FIGURE 3 – En abscisse, le nombre d'instance de jeux en ordonnée, le pourcentage de victoires de MAC-UCB contre UCT avec 30 secondes par coup sur *Backgammon*.

Les figures 2 et 3 présentent, respectivement pour *Awale* et *Backgammon*, l'évolution du pourcentage de victoires de MAC-UCB contre UCT avec 30 secondes par coup, en faisant évoluer le nombre d'instances jusqu'à 5,000. Pour le jeu déterministe *Awale*, la performance est pratiquement stationnaire au bout de 1,000 instances. Ici, seul le filtrage réalisé par MAC sur la partie CSP des réseaux stochastiques a un effet sur cette performance. A l'opposé, pour le jeu stochastique *Backgammon*, la performance de MAC-UCB s'améliore sensiblement, avec une augmentation de victoires de 10% au bout de 4,000 parties. Cette amélioration peut s'expliquer par l'augmentation du nombre d'élagages probabilistes et déterministes induits par UCB, dont le cache grossit progressivement. Un phénomène similaire est observé pour les autres jeux stochastiques.

La figure 4 montre le pourcentage de victoires de MAC-UCB contre UCT sur le jeu *Can't Stop*, avec différents temps par action (10 à 60 secondes). L'écart de performances entre MAC-UCB et UCT augmente sensiblement avec le temps de décision autorisé ; MAC-UCB obtient 59.9 % de victoires en 10 secondes par coup, et 78.9 % de victoires pour 60 secondes par coup. Cet écart est expliqué par la capacité de MAC à résoudre un nombre de plus en plus grand de μSCSP dans la séquence, ce qui facilite le travail d'exploration d'UCB.

Nous concluons cette partie expérimentale en mettant l'accent sur le dilemme de MAC-UCB entre la résolution (partie MAC) et l'exploration (partie UCB). Dans le cadre de nos expérimentations, nous avons appliqué un ratio de 90 % du temps de décision pour la résolution contre 10 % pour l'échantillonnage. Afin de justifier ce ratio, la figure 5 présente une analyse de sensibilité de MAC-UCB, en décrivant le pourcentage de victoires de MAC-UCB face à UCT pour un ratio variant de 0 % à 100 % pour la partie résolution sur l'ensemble des jeux étu-

3. Notons que ce jeu est en coopération avec les autres joueurs, donc nous présentons la moyennes obtenu pour chaque algorithme

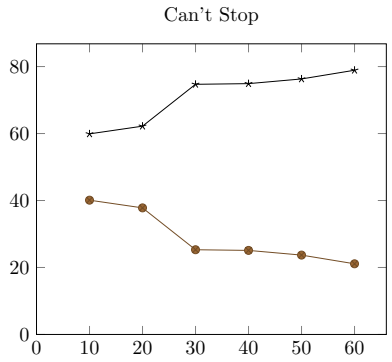


FIGURE 4 – En ordonnée, le pourcentage de victoires de MAC-UCB (×) face à UCT (○) sur *Can't Stop*, et en abscisse les différents temps possibles par coup sur 5000 instances.

diés. De manière régulière, l'optimum est atteint entre 86 et 94 %, ce qui met en lumière l'importance de résoudre les jeux en se focalisant principalement sur leur structure, imposée par les contraintes dures et le choix du seuil.

5 Conclusion

Dans cet article, nous avons exploité un fragment de la modélisation des problèmes de satisfaction par contraintes stochastiques permettant de représenter les jeux à informations complètes et incertaines. Les solutions du réseau stochastique sont interprétées comme des stratégies permettant de jouer à ces jeux. A partir de ce fragment, nous avons proposé un algorithme, MAC-UCB, qui combine résolution et simulation en utilisant à la fois une technique de programmation par contraintes pour la résolution (MAC) et une technique d'apprentissage stochastique pour la simulation (UCB). Testé dans un contexte GGP sur divers jeux avec différents temps par coup, MAC-UCB s'avère, dans la plupart des cas, être plus performant qu'UCT, la référence dans le domaine des jeux à information complètes et incertaines.

Ce travail ouvre la voie à de nombreuses perspectives de recherche. Notamment, afin d'optimiser la résolution, nous souhaitons exploiter les symétries présentes dans les μSCSP obtenus et les méthodes d'arc consistances dédiées aux SCSP (ex : [2]). De plus, il serait intéressant d'exploiter une borne de confiance plus adaptée aux problèmes de jeux (souvent

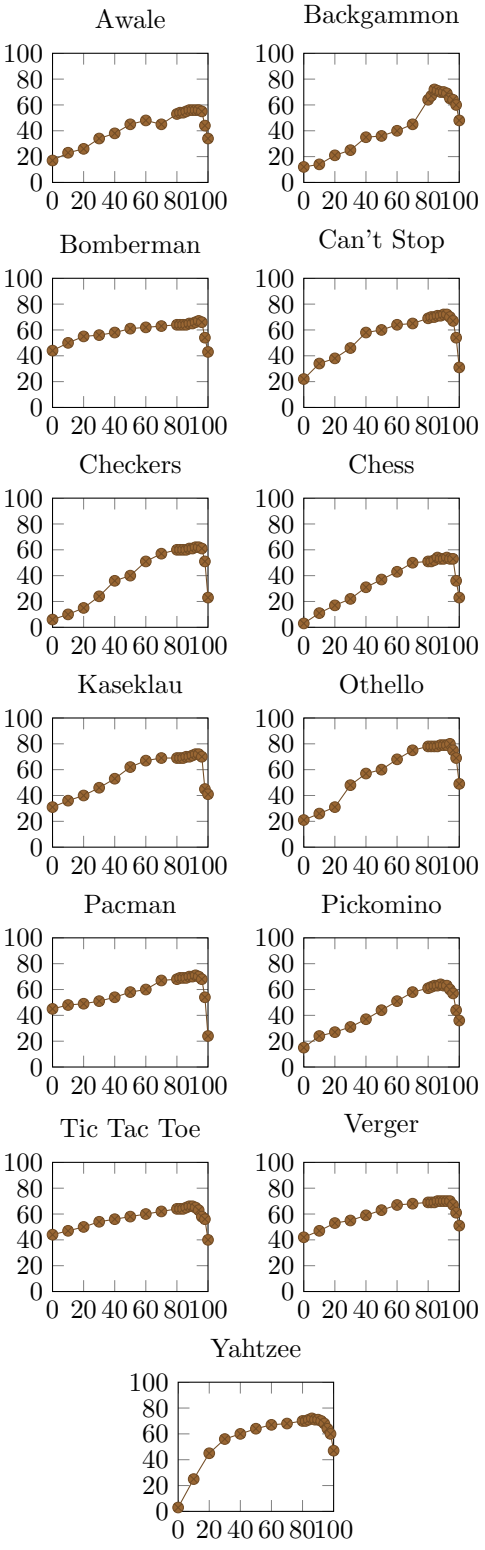


FIGURE 5 – Analyse de sensibilité entre résolution et simulation de MAC-UCB. En ordonnée, le pourcentage de victoires de MAC-UCB vs. UCT et en abscisse, le pourcentage de résolution durant les temps d'action.

compétitifs) afin d'obtenir de meilleures heuristiques. A plus long terme, nous souhaiterions étendre notre modèle afin d'intégrer les jeux à informations incomplètes.

Références

[1] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3) :235–256, 2002.

[2] Thanasis Balafoutis and Kostas Stergiou. Algorithms for stochastic CSPs. In *Proc. of CP'06*, pages 44–58, 2006.

[3] Christian Bessiere and Guillaume Verger. Strategic constraint satisfaction problems. In *Proc. of CP'06 Workshop on Modelling and Reformulation*, pages 17–29, 2006.

[4] James Edmond Clune, III. *Heuristic evaluation functions for general game playing.* PhD thesis, University of California, Los Angeles, USA, 2008. Adviser-Korf, Richard E.

[5] R. Debruyne and C. Bessiere. Some practical filtering techniques for the constraint satisfaction problem. In *Proc. of IJCAI'97*, pages 412–417. Springer, 1997.

[6] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *Proc. of AAAI'08*, pages 259–264, 2008.

[7] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing : Overview of the aaai competition. *AAAI Magazine*, 26(2) :62–72, 2005.

[8] Ian P. Genta, Peter Nightingalea, Andrew Rowleya, and Kostas Stergiou. Solving quantified constraint satisfaction problems. *Artificial Intelligence*, 172(6-7) :738–77, 2008.

[9] Frédéric Koriche, Sylvain Lagrue, Éric Piette, and Sébastien Tabary. Compiling strategic games with complete information into stochastic csps. In *AAAI workshop on Planning, Search, and Optimization (PlanSOpt-15)*, 2015.

[10] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General game playing : Game description language specification. Technical report, 2008.

[11] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8 :99–118, 1977.

[12] A.K. Mackworth. On reading sketch maps. In *Proc. of IJCAI'77*, page 598–606. Springer, 1977.

[13] Maximilian Möller, Marius Thomas Schneider, Martin Wegner, and Torsten Schaub. Centurio, a general game player : Parallel, Java- and ASP-based. *Künstliche Intelligenz*, 25(1) :17–24, 2011.

[14] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning : Theory & Practice*. Morgan Kaufmann, 2004.

[15] Thi-Van-Anh Nguyen, Arnaud Lallouet, and Lucas Bordeaux. Constraint games : Framework and local search solver. In *Proc. of ICTAI'13*, pages 8–12, 2013.

[16] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proc. of CP'94*, pages 10–20. Springer, 1994.

[17] D. Sabin and E.C. Freuder. Understanding and improving the mac algorithm. In *Proc. of CP'97*, pages 167–181. Springer, 1997.

[18] Nathan R. Sturtevant. An analysis of uct in multi-player games. *ICGA Journal*, 31(4) :195–208, 2008.

[19] Michael Thielscher. Flux : A logic programming method for reasoning agents. *Theory Pract. Log. Program.*, 5(4-5) :533–565, 2005.

[20] Michael Thielscher. A general game description language for incomplete information games. In *Proc. of AAAI'10*, pages 994–999, 2010.

[21] Toby Walsh. Stochastic constraint programming. In *Proc. of ECAI'02*, pages 111–115, 2002.

CoQuiAAS : Applications de la programmation par contraintes à l'argumentation abstraite

Jean-Marie Lagniez Emmanuel Lonca Jean-Guy Mailly

CRIL – Université d'Artois, CNRS
Lens, France
{lagniez,lonca,mailly}@cril.univ-artois.fr

Résumé

L'argumentation est aujourd'hui un des mots-clés prépondérants en ce qui concerne l'intelligence artificielle. Elle a notamment sa place dans des thématiques comme les systèmes multi-agent, où elle permet la mise en place de protocoles de dialogue (à base de persuasion, de négociation,...) ou l'analyse de discussions en ligne ; mais aussi dans le cas où un unique agent doit raisonner en présence d'informations conflictuelles (inférence en présence d'incohérence, mesures d'incohérence). Ce cadre très riche offre la possibilité de raisonner suivant un nombre important de sémantiques et propose deux politiques d'inférence pour chaque sémantique.

D'autre part, les travaux réalisés sur SAT ces dernières années, et de façon générale dans le domaine de la programmation par contraintes, ont permis d'obtenir des approches efficaces en pratique pour traiter des problèmes dont la complexité théorique est élevée.

Les besoins d'applications efficaces pour effectuer les tâches de raisonnement usuelles à partir d'un système d'argumentation, ainsi que la puissance des techniques modernes de programmation par contraintes, nous ont amené à étudier l'encodage des sémantiques usuelles en argumentation abstraite dans des formalismes logiques, afin de pouvoir utiliser diverses approches de la programmation par contraintes pour développer une bibliothèque logicielle de raisonnement argumentatif. La bibliothèque que nous proposons présente l'avantage d'être générique et aisément adaptable. Après une présentation de la conception de notre logiciel, nous étudions expérimentalement notre approche sur un ensemble de sémantiques et de tâches d'inférence usuelles.

1 Introduction

Un système d'argumentation abstrait [15] est un graphe orienté dont les nœuds représentent des entités abstraites appelées *arguments* et dont les arêtes représentent des *attaques* entre ces arguments. Ce cadre simple et élégant est utilisé aussi bien dans des traitements concernant un seul agent que dans des scénarios multi-agent. En ce qui concerne le cas d'un unique agent, celui-ci peut par exemple utiliser un système d'argumentation pour raisonner à partir d'informations contradictoires, ce qui peut entre autre lui permettre de construire un système d'argumentation à partir d'une base de connaissances incohérente pour en tirer des conséquences non triviales [8]). Dans un cadre multi-agent, on peut utiliser l'argumentation pour modéliser des dialogues entre plusieurs agents [3] ou pour analyser une discussion en ligne entre utilisateurs de réseaux sociaux [24]. Le sens d'un tel graphe est déterminé par une *sémantique d'acceptabilité*, qui indique quelles propriétés un ensemble d'arguments doit satisfaire pour que cet ensemble soit considéré comme une « solution » acceptable du problème ; on appelle alors cet ensemble une *extension*.

À l'heure actuelle, une des tendances fortes dans la communauté de l'argumentation est de développer des approches logicielles pour calculer diverses tâches d'inférence à partir d'un système d'argumentation pour les sémantiques usuelles (voir `http://argumentationcompetition.org/2015` pour plus de détails). Pour une sémantique donnée, les requêtes usuelles concernent majoritairement le calcul d'une extension, de toutes les extensions, ou bien encore l'acceptation crédule (respectivement sceptique) d'un argument particulier, c'est-à-dire son appartenance à au moins une (respectivement chaque) extension du système d'argumentation. Il est connu que pour la plupart des couples composés d'une sémantique et d'une requête usuelles, la complexité théorique des problèmes est élevée [14, 17]. Ces tâches nécessitent de ce fait

de disposer d'outils efficaces en pratique pour pouvoir être calculées en temps raisonnable. Pour parvenir à calculer efficacement les extensions d'un système d'argumentation, ainsi que pour parvenir à raisonner sur l'acceptation crédule ou sceptique d'un argument en temps raisonnable, nous proposons dans cet article d'utiliser diverses techniques liées à la programmation par contraintes, domaine dont les approches proposées apportent des solutions efficaces en terme de raisonnement logique pour les problèmes combinatoires. Nous nous intéressons tout particulièrement dans cet article à la logique propositionnelle et aux formalismes dérivés de cette dernière. Plus précisément, nous proposons des encodages sous forme normale conjonctive pour des problèmes de décision du premier niveau de la hiérarchie polynomiale, mais aussi des encodages dans le formalisme *Partial Max-SAT*; ces encodages nous permettent de répondre à des requêtes concernant quatre sémantiques et quatre requêtes usuelles. Nous tirons ensuite parti de ces encodages pour résoudre les problèmes de notre étude en utilisant des logiciels et approches de l'état de l'art ayant démontré leur efficacité pratique.

Nous avons encodé ces différentes approches pour le raisonnement à partir d'un système d'argumentation dans une bibliothèque logicielle appelée CoQuiAAS. Le but de notre bibliothèque est non seulement de fournir des algorithmes de base permettant de gérer les principales requêtes pour les principales sémantiques, mais aussi de fournir un framework évolutif dans lequel l'ajout de nouveaux paramètres (requête, sémantique) ou la réalisation de nouveaux algorithmes de résolution pour des problèmes déjà gérés est aisée.

Dans cet article, nous présentons en premier lieu les notions de base concernant l'argumentation abstraite et les problèmes vers lesquels nous réduisons nos requêtes, à savoir le problème SAT (recherche d'un modèle dans une formule sous forme normale conjonctive) et le problème de la recherche de sous-ensembles maximaux cohérents (MSS) dans des instances dites *Partial Max-SAT*. Nous détaillons à la suite de cette présentation les encodages que nous avons employés afin de réduire les problèmes d'argumentation considérés aux problèmes de vérification de cohérence ou de recherche de sous-ensemble maximal cohérent. Nous présentons ensuite dans la section 4 la façon dont nous avons conçu notre bibliothèque, CoQuiAAS. Enfin, nous donnons des résultats expérimentaux concernant les encodages que nous proposons en section 5, et nous comparons d'un point de vue conception notre bibliothèque aux logiciels existants capables de calculer des requêtes faisant parti de notre étude.

2 Préliminaires formels

2.1 Logique propositionnelle

L'alphabet de la logique propositionnelle est composé d'un ensemble de variables booléennes et d'un ensemble de trois opérateurs usuels : l'opérateur de négation $\neg$ (unaire), l'opérateur de conjonction $\wedge$ (binaire) et l'opérateur de disjonction $\vee$ (binaire), chacun de ces opérateurs connectant des formules entre elles (les variables propositionnelles étant elle-même des formules). Dans la mesure où une formule propositionnelle est construite de manière inductive (puisque les connecteurs s'appliquent sur des formules), elle peut de ce fait être vue comme un arbre. Étant donnée une affectation de l'ensemble des variables propositionnelles (appelée interprétation), une formule propositionnelle est évaluée à *vrai* si et seulement si le nœud racine de la formule est évalué à *vrai*. Pour connaitre la valeur de ce nœud, il suffit de procéder au calcul de chacun des nœud dans l'ordre topologique inverse; en effet la valeur des feuilles est connue (puisque les feuilles sont les nœuds correspondant aux variables), et la valeur des nœuds internes peut être déterminée en fonction de la sémantique attachée aux connecteurs qui leurs sont associés : un nœud négation ($\neg$) a pour valeur *vrai* si et seulement si son fils a la valeur *faux*, un nœud $\wedge$ (respectivement $\vee$) a la valeur *vrai* si et seulement si ses deux (respectivement un de ses deux fils) fils ont (respectivement a) pour valeur *vrai*. Lorsqu'une interprétation donne à une formule la valeur *vrai*, on dit que c'est un *modèle* de la formule. Par convention, on représente une interprétation par l'ensemble des variables affectées à *vrai* par cette interprétation. Usuellement, on définit en sus des trois opérateurs présentés les connecteurs d'implication ($\Rightarrow$, tel que $a \Rightarrow b \equiv \neg a \vee b$) et d'équivalence ($\Leftrightarrow$, tel que $a \Leftrightarrow b \equiv (a \Rightarrow b) \wedge (b \Rightarrow a)$). Lorsqu'une formule propositionnelle Φ est équivalente à une conjonction $\varphi_1 \wedge \cdots \wedge \varphi_n$, on peut représenter Φ comme un ensemble $\{\varphi_1, \ldots, \varphi_n\}$.

Dans notre étude, nous traitons d'encodages en formules NNF (*Negation Normal Form*), c'est-à-dire des formules propositionnelles dans lesquelles les négations ne portent que sur des variables (nœuds terminaux). Cependant, les prouveurs SAT n'étant capables que de gérer des formules CNF (*Conjunctive Normal Form*), une étape de traduction de formule NNF vers CNF est nécessaire entre les encodages présents dans l'état de l'art et ceux que nous utilisons de manière effective; ceci n'influe cependant pas la généralité de nos approches dans la mesure où, pour toute formule propositionnelle, on peut déterminer en temps polynomial une formule CNF équivalente.

Les formules CNF correspondent aux conjonctions

de disjonction de littéraux (un littéral est une variable propositionnelle potentiellement niée), c'est-à-dire les formules écrites sous la forme $\wedge_{i=1}^{n}(\vee_{j=1}^{n_i} l_{i,j})$. Ces formules sont intéressantes dans la mesure où une disjonction de littéraux (appelée *clause*) permet de représenter aisément une contrainte ; une formule CNF est ainsi en fait un ensemble de contraintes, et un modèle d'une formule CNF est alors une affectation des variables telle que toutes les contraintes d'un problème soient satisfaites.

Bien que ce formalisme soit adapté à la représentation d'un problème, la recherche d'un modèle pour une formule CNF est théoriquement complexe (NP-difficile [13]). Cependant, des logiciels (appelés *prouveurs SAT*, ou encore *solveurs SAT*) sont dédiés à la résolution de tels problèmes et permettent d'en résoudre de plus en plus imposant [9]. Il existe néanmoins des problèmes pour lesquels il n'existe aucun modèle, c'est-à-dire aucune affectation complète des variables telle que l'ensemble des contraintes soit satisfait. On peut dans ce cas chercher à déterminer une interprétation qui maximise le nombre de contraintes résolues : on nomme ce problème *Max-SAT* [9]. On peut généraliser ce problème en donnant un poids à chacune des contraintes (on cherche alors à maximiser la somme des poids des contraintes satisfaites) ; il s'agit alors du problème *Weighted Max-SAT*. Si des contraintes ont un poids infini (elles doivent impérativement être satisfaites), on passe alors dans le cadre des problèmes *Partial Max-SAT* et *Weighted Partial Max-SAT*.

Déterminer une solution à un problème Max-SAT permet donc de déterminer un ensemble de contraintes de la formule initiale qui est cohérent, tel que l'ajout de n'importe quelle autre contrainte issue du problème de départ rend ce nouvel ensemble incohérent [25] ; un sous-ensemble de contraintes d'une formule ayant cette propriété est un *sous-ensemble maximal cohérent* (MSS) [26]. Étant donné l'ensemble de contraintes φ d'un problème et ψ un ensemble de contraintes formant un MSS de φ, on dit que $\overline{\psi} = \varphi \setminus \psi$ est un *coMSS* (ou *MCS*) de la formule [25]. Notons toutefois que les solutions optimales pour le problème Max-SAT ne forment qu'un sous-ensemble des MSS d'une formule.

Il est intéressant de noter qu'en ce qui concerne les algorithmes développés en programmation par contraintes pour la résolution de problème Max-SAT ou pour la recherche de MSS, l'approche utilisée consiste généralement à utiliser comme boite noire un prouveur SAT classique basé sur l'interface incrémentale de Minisat [4, 20, 23] de manière successive sur différents problèmes de test de cohérence. En ce qui concerne le calcul de MSS par le biais d'un tel prouveur, on peut par exemple citer l'algorithme BLS [5] ou plus récemment l'algorithme CMP [22] ; notons toutefois que certains logiciels dévolus au calcul de MSS utilisent à cette fin un prouveur Max-SAT comme boite noire [25].

2.2 Argumentation abstraite

Plusieurs modèles ont été proposés pour formaliser l'argumentation. Nous travaillons avec l'un des plus populaires, le cadre proposé par Dung [15].

Définition 1. *Un* système d'argumentation *est un graphe orienté $F = \langle A, R\rangle$ où A désigne un ensemble fini d'objets appelés des* arguments *et $R \subseteq A \times A$ une relation appelée* relation d'attaque.

La signification intuitive de la relation d'attaque est celle qui est présente lorsque l'on débat avec quelqu'un. Si un premier argument a_1 est avancé, sans opposition, rien *a priori* n'empêche de considérer que a_1 est vrai. Par contre, si quelqu'un avance un argument a_2 (*a priori* acceptable) qui attaque a_1, alors a_1 ne peut pas être accepté, à moins qu'il ne soit ensuite défendu. Cette notion intuitive de défense peut être formalisée :

Définition 2. *Soient $F = \langle A, R\rangle$ un système d'argumentation et $a_1, a_2, a_3 \in A$ trois arguments.*

- *L'argument a_1 est* attaqué *par l'argument a_2 dans F si et seulement si $(a_2, a_1) \in R$.*
- *On dit alors que l'argument a_3* défend *a_1 contre a_2 dans F si et seulement si $(a_3, a_2) \in R$.*

On généralise ces notions à l'attaque et la défense d'un argument par un ensemble d'argument $E \subseteq A$.

- *L'argument a_1 est* attaqué *par l'ensemble d'arguments E dans F si et seulement si $\exists a_i \in E$ tel que $(a_i, a_1) \in R$.*
- *L'ensemble d'arguments $E \subseteq A$* défend *a_1 contre a_2 dans F si et seulement si E attaque a_2.*

Par exemple, dans le système d'argumentation décrit en Figure 1, l'argument a_1 attaque l'argument a_2, et a_2 se défend lui-même contre cette attaque.

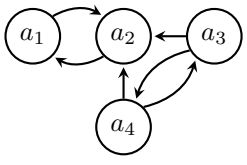


FIGURE 1 – Un exemple de système d'argumentation

Lorsque l'on raisonne avec un système d'argumentation, la première tâche est habituellement de déterminer quels ensembles d'arguments peuvent être acceptés conjointement. Différentes propriétés peuvent être exigées d'un ensemble d'arguments pour qu'il soit

considéré comme une « solution » raisonnable du système d'argumentation. Parmi ces propriétés, deux en particulier sont exigées par la plupart des sémantiques usuelles :

— l'*absence de conflit* : $E \subseteq A$ est sans conflit dans F si et seulement si $\nexists a_i, a_j \in E$ tels que $(a_i, a_j) \in R$;

— l'*admissibilité* : un ensemble sans conflit $E \subseteq A$ est admissible si et seulement si $\forall a_i \in E$, E défend a_i contre tous ses attaquants.

L'admissibilité et l'absence de conflit sont à la base des quatre sémantiques usuelles de Dung :

Définition 3. *Soit $F = \langle A, R \rangle$ un système d'argumentation.*

— *Un ensemble sans conflit $E \subseteq A$ est une* extension complète *de F si et seulement si E contient tous les arguments que défend E.*

— *Un ensemble $E \subseteq A$ est une* extension préférée *de F si et seulement si E est un élément maximal selon $\subseteq$ parmi les extensions complètes de F.*

— *Un ensemble sans conflit $E \subseteq A$ est une* extension stable *de F si et seulement si E attaque tous les arguments qui n'appartiennent pas à E.*

— *Un ensemble $E \subseteq A$ est une* extension de base *de F si et seulement si E est un élément minimal selon $\subseteq$ parmi les extensions complètes de F.*

Étant donnée une sémantique σ, on note $Ext_\sigma(F)$ les σ-extensions de F. Nous notons les sémantiques ci-dessus, respectivement, CO, PR, ST et GR.

Nous illustrons en Section 3 ces différentes sémantiques sur le système d'argumentation présenté en Figure 1.

Dung a de plus montré que pour tout système d'argumentation F,

— F admet une et une seule extension de base ;

— F admet au moins une extension préférée ;

— tout extension stable de F est une extension préférée de F ;

— F peut n'admettre aucune extension stable.

Étant donnée une sémantique σ, plusieurs problèmes de décision peuvent être considérés. En premier lieu, il peut être intéressant de déterminer le statut d'acceptation, crédule ou sceptique, d'un argument a. On définit le statut d'un argument a par :

— F accepte a sceptiquement pour la sémantique σ si et seulement si $\forall \varepsilon \in Ext_\sigma(F), a \in \varepsilon$;

— F accepte a crédulement pour la sémantique σ si et seulement si $\exists \varepsilon \in Ext_\sigma(F)$ tel que $a \in \varepsilon$.

De manière évidente, ces deux statuts coïncident pour la sémantique de base, dans la mesure où celle-ci admet une unique extension. Nous notons le problème d'acceptation crédule (respectivement sceptique) DC (respectivement DS).
Un autre problème de décision intéressant est $Exists$, qui désigne le problème consistant à vérifier s'il existe une extension non vide pour une sémantique donnée.

Le complexité pour chacun de ces problèmes de décision est résumée dans la Table 1 (dont les résultats sont issus d'autres publications [14, 17]). $\mathsf{C}-c$ signifie que le problème considéré est complet pour la classe de complexité C.

Sémantique	GR	ST	PR	CO
DC	P	NP$-c$	NP$-c$	NP$-c$
DS	P	coNP$-c$	$\Pi_2^P - c$	P$-c$
$Exist$	P	NP$-c$	NP$-c$	NP$-c$

TABLE 1 – Complexité des problèmes d'inférence pour les sémantiques usuelles

De toute évidence, la complexité du problème $Exist$ est une borne inférieure pour la complexité du calcul d'une extension, tandis que la complexité de DS est une borne inférieure pour la complexité de l'énumération des extensions.

3 Encodages logiques pour l'argumentation abstraite

Il est déjà connu que les sémantiques usuelles des systèmes d'argumentation peuvent être encodées en logique propositionnelle [7]. Nous tirons parti des encodages proposés par Besnard et Doutre pour proposer des approches permettant de calculer les extensions, mais aussi de déterminer si un argument est sceptiquement ou crédulement accepté par un système d'argumentation. Nos encodages sont basés sur un langage propositionnel $\mathcal{L}_A$, défini avec les connecteurs usuels sur l'ensemble de variables propositionnelles $V_A = \{x_{a_i} \mid a_i \in A\}$, x_{a_i} signifiant que l'argument a_i est accepté par le système d'argumentation considéré. Pour raison de lisibilité, nous subsituerons a_i à x_{a_i}.

Rappelons tout d'abord l'encodage de la sémantique stable défini dans [7].

Proposition 1. *Soit $F = \langle A, R \rangle$ un système d'argumentation. $E \subseteq A$ est une extension stable de F si et seulement si E est un modèle de la formule ci-dessous.*

$$\Phi_{st}^F = \bigwedge_{a_i \in A} [a_i \Leftrightarrow (\bigwedge_{a_j \in A | (a_j, a_i) \in R} \neg a_j)]$$

Ainsi, nous pouvons proposer des méthodes très simples pour déterminer les extensions et les statuts d'acceptation des arguments.

Proposition 2. *Soient $F = \langle A, R\rangle$ un système d'argumentation et $a_i \in A$ un argument.*

- *Le calcul d'une extension stable de F revient au calcul d'un modèle de Φ^F_{st}.*
- *L'énumération des extensions stables de F revient à l'énumération des modèles de Φ^F_{st}.*
- *Déterminer si a_i est crédulement accepté par F pour la sémantique stable revient à déterminer si $\Phi^F_{st} \wedge a_i$ est cohérente.*
- *Déterminer si a_i est sceptiquement accepté par F pour la sémantique stable revient à déterminer si $\Phi^F_{st} \wedge \neg a_i$ est incohérente.*

Exemple 1. *Lorsque l'on instancie Φ^F_{st} avec le système d'argumentation présenté en Figure 1, on obtient la formule*

$$(a_1 \Leftrightarrow \neg a_2), (a_2 \Leftrightarrow \neg a_1 \wedge \neg a_3 \wedge \neg a_4), (a_3 \Leftrightarrow \neg a_4), (a_4 \Leftrightarrow \neg a_3)$$

dont les modèles sont $\{a_1, a_3\}$ et $\{a_1, a_4\}$. Les deux extensions stables de F sont donc $Ext_{st}(F) = \{\{a_1, a_3\}, \{a_1, a_4\}\}$.

Comme nous l'avons noté dans la section précédente, les prouveurs SAT ne sont capables que de gérer des formules CNF. En ce qui concerne Φ^F_{st}, l'encodage en formule CNF que nous avons utilisé est le suivant.

$$\Psi^F_{st} = \bigwedge_{a_i \in A} (a_i \vee \bigvee_{a_j \in A|(a_j,a_i)\in R} a_j), \bigwedge_{a_i \in A} [\bigwedge_{a_j \in A|(a_j,a_i)\in R} (\neg a_i \vee \neg a_j)]$$

De façon similaire, Besnard et Doutre ont proposé un encodage en NNF de la sémantique complète.

Proposition 3. *Soit $F = \langle A, R\rangle$ un système d'argumentation. $E \subseteq A$ est une extension complète de F si et seulement si E est un modèle de la formule ci-dessous.*

$$\Phi^F_{co} = \bigwedge_{a_i \in A} [a_i \Rightarrow (\bigwedge_{a_j \in A|(a_j,a_i)\in R} \neg a_j) \wedge (a_i \Leftrightarrow (\bigwedge_{a_j \in A|(a_j,a_i)\in R} \bigvee_{a_k \in A|(a_k,a_j)\in R} a_k))]$$

De la même façon que pour la sémantique stable, nous avons dû traduire cette formule NNF en formule CNF afin de pouvoir utiliser des prouveurs SAT pour nos calculs. En revanche, contrairement au cas précédent, nous avons ajouté des variables auxiliaires P_a définies comme équivalentes à la disjonction des attaquants de l'argument a. Ces variables additionnelles nous permettent d'écrire une formule CNF Ψ^F_{co}, pour laquelle il existe une bijection entre les modèles de Φ^F_{co} et de Ψ^F_{co}, de manière plus élégante qu'en traduisant de manière naïve de NNF vers CNF.

$$\Psi^F_{co} = \bigwedge_{a_i \in A} (\neg a_i \vee \neg P_{a_i}), \bigwedge_{a_i \in A} (a_i \vee \bigvee_{a_j \in A|(a_j,a_i)\in R} \neg P_{a_j}), \bigwedge_{a_i \in A} (\bigwedge_{a_j \in A|(a_j,a_i)\in R} (\neg a_i \vee P_{a_j})), \bigwedge_{a_i \in A} (\neg P_{a_i} \vee \bigvee_{a_j \in A|(a_j,a_i)\in R} a_j), \bigwedge_{a_i \in A} [\bigwedge_{a_j \in A|(a_j,a_i)\in R} (P_{a_i} \vee \neg a_j)]$$

Proposition 4. *Soient $F = \langle A, R\rangle$ un système d'argumentation et $a_i \in A$ un argument.*

- *Le calcul d'une extension complète de F revient au calcul d'un modèle de Φ^F_{co}.*
- *L'énumération des extensions complètes de F revient à l'énumération des modèles de Φ^F_{co}.*
- *Déterminer si a_i est crédulement accepté par F pour la sémantique complète revient à déterminer si $\Phi^F_{co} \wedge a_i$ est cohérente.*
- *Déterminer si a_i est sceptiquement accepté par F pour la sémantique complète revient à déterminer si $\Phi^F_{co} \wedge \neg a_i$ est incohérente.*

Exemple 2. *Lorsque l'on instancie Φ^F_{co} avec le système d'argumentation présenté en Figure 1, on obtient la formule*

$$(a_1 \Rightarrow \neg a_2) \wedge (a_1 \Leftrightarrow a_1 \vee a_3 \vee a_4),$$

$$(a_2 \Rightarrow \neg a_1 \wedge \neg a_3 \wedge \neg a_4) \wedge (a_2 \Leftrightarrow a_2 \wedge a_4 \wedge a_3),$$

$$(a_3 \Rightarrow \neg a_4) \wedge (a_3 \Leftrightarrow a_3),$$

$$(a_4 \Rightarrow \neg a_3) \wedge (a_4 \Leftrightarrow a_4)$$

dont les modèles sont ceux de Φ^F_{st}, auxquels on ajoute $\{a_1\}$ et $\emptyset$. Les extensions complètes de F sont donc $Ext_{co}(F) = \{\{a_1, a_3\}, \{a_1, a_4\}, \{a_1\}, \emptyset\}$.

Les notions de minimalité et maximalité pour $\subseteq$ ne sont pas évidentes à mettre en œuvre directement dans un encodage propositionnel. On définit donc simplement une extension de base (respectivement préférée) comme un modèle minimal (respectivement maximal) pour $\subseteq$ de Φ^F_{co}. On remarque toutefois que l'extension de base d'un système d'argumentation peut être calculée en appliquant la propagation unitaire sur Φ^F_{co}.

Proposition 5. *Soient $F = \langle A, R\rangle$ un système d'argumentation et $a_i \in A$ un argument.*

- *Le calcul de l'extension de base de F revient au calcul des littéraux propagés dans Φ^F_{co}.*
- *Déterminer si a_i est accepté par F pour la sémantique de base revient à déterminer si a_i est propagé dans Φ^F_{co}.*

Exemple 3. *L'application de la propagation unitaire dans Φ^F_{co} définie à l'Exemple 2 donne un ensemble vide : la sémantique de base appliquée à F donne $Ext_{gr}(F) = \{\emptyset\}$.*

Le calcul des extensions préférées de F requiert un encodage légèrement différent. On remarque qu'un modèle maximal de Φ^F_{co} revient au calcul d'un MSS de la formule Φ^F_{pr} définie comme suit :

Proposition 6. *Soit $F = \langle A, R\rangle$ un système d'argumentation. $E \subseteq A$ est une extension préférée de F si et seulement si E est un MSS de la formule pondérée*

$$\Phi^F_{pr} = \{(\Phi^F_{co}, +\infty), (a_1, 1), \cdots, (a_n, 1)\}$$

Nous remarquons tout de même que les requêtes liées à la sémantique préférée ne nécessitent pas toutes d'utiliser l'extraction de MSS d'une instance Partial Max-SAT. En effet, vérifier si un argument est crédulement accepté est NP-complet pour la sémantique préférée. Cela revient en fait exactement à vérifier s'il est crédulement accepté pour la sémantique complète.

Proposition 7. *Soient $F = \langle A, R\rangle$ un système d'argumentation et $a_i \in A$ un argument.*

- *Le calcul d'une extension préférée de F revient au calcul d'un MSS de Φ^F_{pr}.*
- *L'énumération des extensions préférées de F revient à l'énumération des MSS de Φ^F_{pr}.*
- *Déterminer si a_i est crédulement accepté par F pour la sémantique préférée revient à déterminer si $\Phi^F_{co} \wedge a_i$ est cohérente.*
- *Déterminer si a_i est sceptiquement accepté par F pour la sémantique préférée revient à énumérer les MSS de Φ^F_{pr} et vérifier si chacun d'entre eux contient a_i.*

Exemple 4. *Reprenant l'exemple du système d'argumentaiton F donné en Figure 1, Φ^F_{pr} est la formule pondérée*

$$\{(\Phi^F_{co}, +\infty), (a_1, 1), (a_2, 1), (a_3, 1), (a_4, 1)\}$$

dont les MSS sont $\{a_1, a_3\}$ et $\{a_1, a_4\}$, qui sont donc les extensions préférées de F.

4 Conception de la bibliothèque CoQuiAAS

Nous avons fait le choix du langage C++ pour l'implémentation de CoQuiAAS pour ses avantages en terme de paradigme de programmation et d'efficacité. Tout d'abord, l'utilisation du paradigme objet de ce langage nous permet d'envisager une conception élégante pour notre bibilothèque (voir Figure 3), permettant notamment de maintenir et faire évoluer aisément le logiciel. De plus, le langage C++ garantit de bonnes performances du point de vue du temps de calcul, contrairement à d'autres langages orientés objet. Enfin, cela facilite l'intégration de coMSSExtractor, qui est l'outil sous-jacent pour la résolution des problèmes.

Le logiciel coMSSExtractor [22] a été développé comme une surcouche du solveur SAT Minisat [19]. L'API de coMSSExtractor permet à notre logiciel CoQuiAAS d'appeler ses fonctionnalités de solveur SAT et ses fonctionnalités d'extracteur de MSS/coMSS.

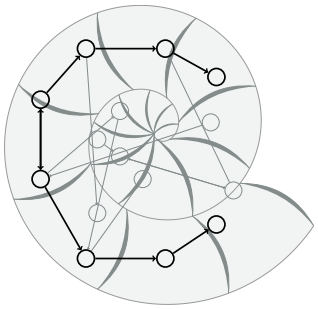


FIGURE 2 – Le logo de CoQuiAAS

Le cœur de notre application est l'interface **Solver**, qui contient les méthodes nécessaires à la résolution des problèmes étudiés. `initProblem` doit être écrite de manière à ce qu'une classe réalisant l'interface **Solver** intègre les données d'entrée, à savoir la formule logique correspondant au système d'argumentation et au problème traité. La méthode `computeProblem` effectue alors les calculs correspondant à la résolution du problème, et `displaySolution` permet d'afficher le résultat au format demandé par la *First International Competition on Computational Models of Argumentation.*

La classe abstraite *SATBasedSolver* (respectivement *CoMSSBasedSolver*) réunit les fonctionnalités et initialisations communes à tous les solveurs basés sur un prouveur SAT (respectivement un extracteur de coMSS), comme par exemple la méthode `hasAModel` qui retourne un booléen indiquant si l'instance SAT produite à partir du problème d'argumentation traité est cohérente ou non. Parmi les classes filles de *SATBasedSolver*, on trouve *DefaultSATBasedSolver* et ses classes filles, qui sont dédiées à l'utilisation de l'API de coMSSExtractor pour employer ses fonctionnalités de solveur SAT héritées de Minisat afin de résoudre les problèmes traités. Si l'utilisateur souhaite faire appel au solveur SAT de son choix au lieu de coMSSExtractor, une option de la ligne de commande

indique à la **SolverFactory** de retourner une instance de la classe *ExternalSATBasedSolver*, elle-aussi classe fille de *SATBasedSolver*, qui est initialisée avec une commande à exécuter afin d'employer tout logiciel externe capable de lire une formule CNF au format DIMACS et de retourner une solution en utilisant le formalisme imposé dans les compétitions de prouveurs SAT. Cette classe permet d'exécuter la commande fournie à CoQuiAAS pour effectuer les calculs liés à la résolution du problème, et peut par exemple permettre de comparer l'efficacité relative de plusieurs solveurs SAT différents sur les instances d'argumentation, dès lors que la sémantique considérée l'autorise. La même conception se retrouve du côté de la classe *CoMSSBasedSolver*, qui peut être instanciée *via* le solveur par défaut *DefaultCoMSSBasedSolver*, qui fait appel aux fonctions de l'API de CoMSSExtractor, ou *via* la classe *ExternalCoMSSBasedSolver* pour utiliser un logiciel tiers dont les entrées et sorties correspondent à celles de coMSSExtractor, pour les couples requête/sémantique gérés.

Notre conception est suffisamment souple pour permettre une évolution aisée de la bibliothèque CoQuiAAS. Il est par exemple simple de créer un prouveur basé sur l'API d'un prouveur SAT autre que coMSSExtractor : il suffit de créer une nouvelle classe **MySolver** qui hérite de *SATBasedSolver* (et donc de l'interface mère de tous les prouveurs, **Solver**), puis d'implémenter les méthodes abstraites de cette classe, à savoir les fonctions `initProblem`, `hasAModel`, `getModel` et `addBlockingClause`. Il est aussi possible, par exemple, d'hériter directement de la classe **Solver** et d'écrire ses trois méthodes `initProblem`, `computeProblem` et `displaySolution` pour créer un prouveur quelconque. Par exemple, si l'on souhaite développer des solveurs de problèmes d'argumentation utilisant le cadre des CSP, en se basant sur des encodages comme ceux présentés dans [2], il suffit d'ajouter une nouvelle classe *CSPBasedSolver* qui implémente l'interface **Solver**, et de reproduire le processus qui a mené à la conception des solveurs basés sur SAT, en se basant cette fois-ci sur l'API d'un solveur CSP (ou sur un solveur CSP externe).

Une fois le prouveur rédigé, il suffit ensuite de relier une option de la ligne de commande à l'utilisation de ce prouveur, en mettant à jour pour cela la méthode `getSolverInstance` de la **SolverFactory**, qui dispose parmi ses paramètres de l'ensemble des arguments de la ligne de commande de CoQuiAAS, via la map `opt`. On peut par exemple relier le paramètre `-solver MySolver` à l'utilisation de la classe **MySolver** dédiée au nouveau solveur en ajoutant le simple code suivant.

```
if(!opt["-solver"].compare("MySolver"))
    return new MySolver(...);
```

De la façon dont l'interface **Solver** est conçue, il est supposé qu'un solveur soit consacré à un unique problème pour une unique sémantique. Il est ainsi possible d'implémenter une classe dédiée à une sémantique et un problème particuliers, utilisant un algorithme adapté à ceux-ci. Par exemple, [10] décrit une procédure qui permet de déterminer si un argument donné est dans l'extension de base d'un système d'argumentation. On peut envisager d'implémenter une classe **GroundedDiscussion** qui implémente elle-même l'interface **Solver** afin de résoudre le problème d'inférence sceptique sous la sémantique de base.

Ce comportement de CoQuiAAS n'empêche cependant pas l'implémentation de prouveurs capables de résoudre plusieurs requêtes d'une même sémantique, à partir du moment où la **SolverFactory** redirige les différents problèmes à traiter vers la même classe. Ainsi, étant donnée la possibilité d'utiliser, pour une sémantique donnée, différentes approches toutes basées sur SAT (ou les coMSS) quel que soit le problème traité, nous avons simplifié la conception de nos solveurs en utilisant une classe par sémantique. Par exemple, la méthode `computeProblem` est implémentée de la façon décrite dans l'Algorithme 1 dans la classe **CompleteSemanticSolver**.

Algorithme 1 : `computeProblem`

Données : Un système d'argumentation F, un problème P, un argument a_i
suivant P **faire**
  cas où *Calcul d'une extension*
    computeOneExtension(F)
  cas où *Énumération des extensions*
    computeAllExtensions(F)
  cas où *Acceptation crédule*
    checkCredulousAcceptance(F,a_i)
  cas où *Acceptation sceptique*
    checkSkepticalAcceptance(F,a_i)

Notons que la valeur du paramètre a_i est ignorée lorsque le problème considéré est le calcul d'une extension ou l'énumération des extensions.

5 Expérimentations

Nous avons mené nos expérimentations sur les jeux d'instances fournis par les organisateurs de la *First International Competition on Computational Models of Argumentation* afin d'éprouver les solveurs en amont de la dite compétition. Le premier jeu d'instance se

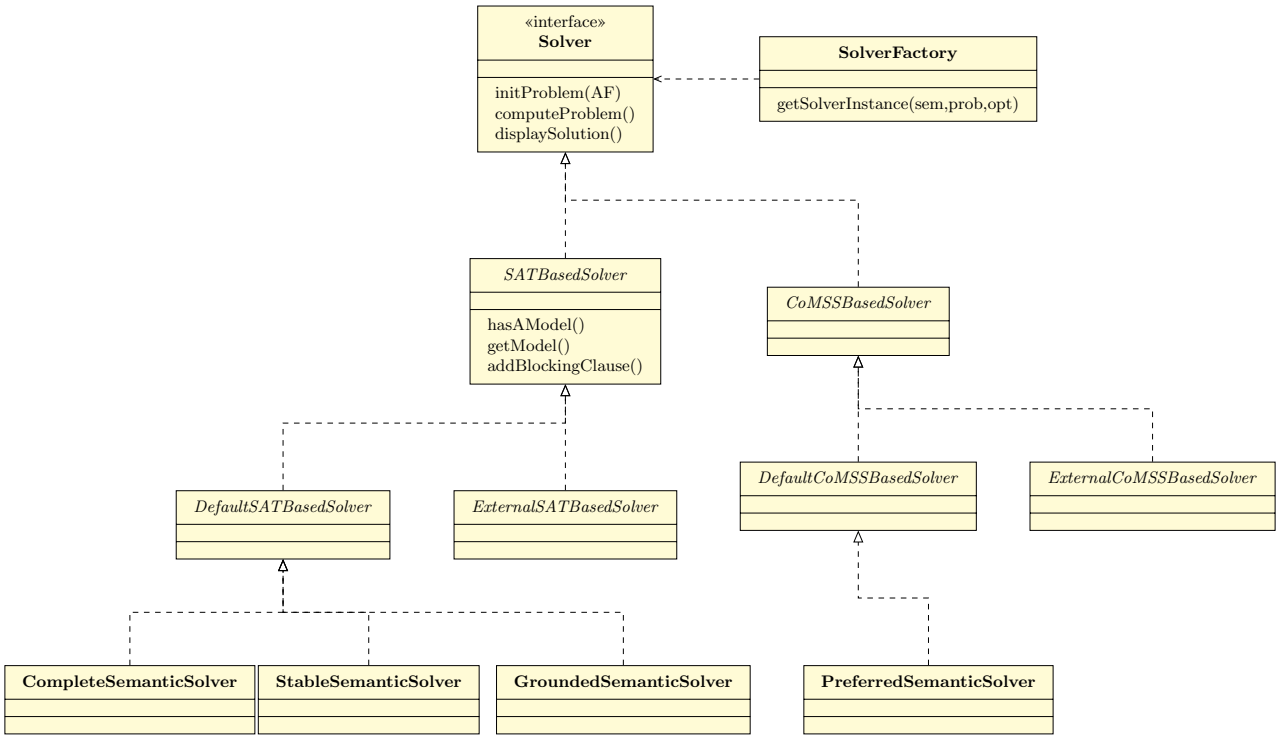


FIGURE 3 – Diagramme de classe simplifié de la partie « solveur » de CoQuiAAS

décompose en une famille de 20 instances dites `real`, dont le nombre d'arguments va de 5000 à 100000, et 79 instances aléatoires dont le nombre d'arguments varie entre 20 et 1000. Le deuxième jeu d'instances comporte des instances aléatoires dont le nombre d'arguments varie entre 200 et 400. CoQuiAAS a été exécuté sur des machines équipées de processeurs Intel Xeon cadencés à 3.00 GHz associés à 2 Go de mémoire vive, dont le système d'exploitation est la distribution GNU/Linux CentOS 6.0 (64 bits).

Nous avons en priorité étudié l'efficacité pratique de notre méthode sur les problèmes d'énumération, étant donné que le temps pour énumérer les extensions d'un système d'argumentation est une borne supérieure du temps requis pour les autres tâches que nous avons implémentées. Les résultats sont donnés en Table 2. Nous avons agrégé les temps par famille d'instances, et présentons ici les temps moyens, arrondis à 10^{-2} près. Le symbole – indique que l'intégralité de la famille a atteint le *timeout* fixé à 900 secondes. Les autres familles ont été intégralement résolues.

Nos expérimentations montrent l'efficacité de notre approche sur les instances de la compétition. Seules les instances `rdm1000` ont atteint le *timeout* sans être résolues, pour les sémantiques stable, complète et préférée. Les temps moyens pour la résolution des instances `XXX300` avec ces trois sémantiques sont nettement plus élevés que les temps moyens requis pour les instances `XXX400` sur les mêmes sémantiques. Cela s'explique par la présence de quelques instances particulièrements difficiles dans cette famille. Certaines d'entre elles nécessitent plusieurs dizaines de secondes, et même jusque 280 secondes pour la sémantique complète et 653 secondes pour la sémantique préférée ; cependant la grande majorité des instances de cette famille est tout de même résolue très rapidement (41 instances sont résolues en strictement moins d'une seconde avec la sémantique complète, et 37 instances pour la sémantique préférée).

Famille	#Inst.	Gr	St	Pr	Co
`rdm20`	25	< 0.01	0.01	< 0.01	< 0.01
`rdm50`	24	< 0.01	< 0.01	< 0.01	< 0.01
`rdm200`	24	< 0.01	0.5	5.32	1.57
`rdm1000`	6	0.25	–	–	–
`real`	20	7.25	7.51	8.55	6.88
`XXX200`	4	< 0.01	0.08	0.04	0.03
`XXX300`	64	0.02	12.53	34.8	21.36
`XXX400`	22	0.01	0.12	0.13	0.08

TABLE 2 – Temps moyen par famille pour énumérer les extensions pour les différentes sémantiques

6 Travaux connexes

Plusieurs approches similaires ont été développées ces dernières années. ASPARTIX [21], tout d'abord, propose une implémentation basée sur des techniques ASP pour calculer les extensions d'un système d'argumentation, pour un grand nombre de sémantiques. Il ne permet par contre pas de travailler avec des requêtes comme l'acceptation crédule et l'acceptation sceptique d'un argument. CEGARTIX [18], basé sur SAT, se concentre sur les requêtes dont la complexité est au second niveau de la hiérarchie polynomiale. Bien que performante, la version actuelle de ce logiciel est nettement moins générale que notre bibliothèque, et qui permet déjà de travailler avec toutes les requêtes et sémantiques usuelles, et qui peut être aisément étendue pour travailler avec les autres sémantiques. ArgSemSAT [12], enfin, est aussi basé sur SAT et permet l'énumération des extensions pour les sémantiques usuelles. Pour autant que nous sachions, aucun de ces logiciels ne permet l'intégration aisée d'autres types de solveurs de contraintes.

7 Conclusion

Dans cet article, nous présentons notre approche basée sur SAT et sur l'extraction de MSS dans le formalisme Partial Max-SAT pour résoudre les problèmes d'inférence les plus courants en argumentation abstraite. Nous mettons en avant la conception de notre bibliothèque CoQuiAAS, qui a été développée de façon à être aisément maintenable et évolutive. Une première version de notre logiciel est d'ors et déjà disponible en ligne à l'adresse `http://www.cril.univ-artois.fr/coquiaas`.

Plusieurs pistes sont envisagées pour de futurs travaux. Tout d'abord, concernant l'inférence à partir d'un système d'argumentation abstrait, il existe un certain nombre de sémantiques qui ont été proposées après les quatre sémantiques « classiques » de Dung. Développer une approche pour ces sémantiques est un défi particulièrement intéressant, notamment la sémantique semi-stable [11] et la sémantique stage [27], pour lesquelles même l'acceptation crédule est au deuxième niveau de la hiérarchie polynomiale. Toujours dans le cadre de Dung, nous savons que lorsque certaines propriétés sont satisfaites par le graphe, certaines sémantiques coïncident. Par exemple, si le système d'argumentation est dit « cohérent » (suivant la définition donnée dans [15]), alors la sémantique stable et la sémantique préférée coïncident. Ainsi, il n'est pas nécessaire dans ce cas d'utiliser l'approche à base de MSS pour calculer les extensions préférées, il est possible d'utiliser l'approche pour la sémantique stable, qui est basée sur SAT et est beaucoup plus efficace. De même, il existe une propriété permettant de déterminer si toutes les sémantiques coïncident, ce qui permet de se limiter à la sémantique de base pour raisonner. Nous souhaitons donc développer des méthodes de *pre-processing* afin de permettre de détecter les graphes pour lesquels il est possible d'utiliser une approche dont la complexité est moins élevée.
Nous envisageons aussi d'étendre notre travail aux diverses extensions du cadre de Dung, tels les Weighted Argumentation Frameworks [16], les Preference-based Argumentation Frameworks [1] ou les Value-based Argumentation Frameworks [6].
Enfin, du point de vue de l'expérience de l'utilisateur, il serait très intéressant d'avoir un moyen de visualiser les systèmes d'argumentation et les résultats des requêtes effectuées sur ces systèmes. Nous envisageons donc de développer une interface graphique, basée sur le moteur de calculs de CoQuiAAS, pour améliorer la qualité des interactions entre l'utilisateur et le système.

Remerciements

Nous tenons à remercier les relecteurs pour leur travail et les commentaires qu'ils ont exprimés.
Ce travail a bénéficié d'une aide de l'Agence Nationale de la Recherche portant la référence ANR-13-BS02-0004 dans le cadre du projet AMANDE.
Ce travail est financé en partie par le conseil régional Nord-Pas de Calais et le programme FEDER.

Références

[1] Leila Amgoud and Claudette Cayrol. A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34(1-3) :197–215, 2002.

[2] Leila Amgoud and Caroline Devred. Argumentation frameworks as constraint satisfaction problems. *Annals of Mathematics and Artificial Intelligence*, 69(1) :131–148, 2013.

[3] Leila Amgoud and Nabil Hameurlain. An argumentation-based approach for dialog move selection. In *3rd International Workshop on Argumentation in Multi-Agent Systems, ArgMAS 2006*, pages 128–141, 2006.

[4] Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving glucose for incremental SAT solving with assumptions : Application to MUS extraction. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing - SAT 2013*, pages 309–317, 2013.

[5] James Bailey and Peter J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using

hitting set dualization. In *Proceedings of the 7th International Symposium on Practical Aspects of Declarative Languages, PADL 2005*, pages 174–186, 2005.

[6] Trevor J. M. Bench-Capon. Value-based argumentation frameworks. In *Proceedings of the 9th International Workshop on Non-Monotonic Reasoning, NMR 2002*, pages 443–454, 2002.

[7] Philippe Besnard and Sylvie Doutre. Checking the acceptability of a set of arguments. In *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning, NMR 2004*, pages 59–64, 2004.

[8] Philippe Besnard and Anthony Hunter. *Elements of Argumentation*. MIT Press, 2008.

[9] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[10] Martin Caminada and Mikolaj Podlaszewski. Grounded semantics as persuasion dialogue. In *Proceedings of the 4th International Conference on Computational Models of Argument, COMMA 2012*, pages 478–485, 2012.

[11] Martin W. A. Caminada, Walter Alexandre Carnielli, and Paul E. Dunne. Semi-stable semantics. *Journal of Logic and Computation*, 22(5) :1207–1254, 2012.

[12] Federico Cerutti, Massimiliano Giacomin, and Mauro Vallati. Argsemsat : Solving argumentation problems using SAT. In *Proceedings of the 5th International Conference on Computational Models of Argument, COMMA 2014*, pages 455–456, 2014.

[13] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[14] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Symmetric argumentation frameworks. In *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2005*, pages 317–328, 2005.

[15] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n-person games. *Artificial Intelligence*, 77(2) :321–357, 1995.

[16] Paul E. Dunne, Anthony Hunter, Peter McBurney, Simon Parsons, and Michael Wooldridge. Weighted argument systems : Basic definitions, algorithms, and complexity results. *Artificial Intelligence*, 175(2) :457–486, 2011.

[17] Paul E. Dunne and Michael Wooldridge. Complexity of abstract argumentation. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, chapter 5, pages 85–104. Springer, 2009.

[18] Wolfgang Dvořák, Matti Järvisalo, Johannes P. Wallner, and Stefan Woltran. CEGARTIX : A SAT-Based Argumentation System. Pragmatics of SAT Workshop, PoS 2012, 2012.

[19] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing - SAT 2003*, pages 502–518, 2003.

[20] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electr. Notes Theor. Comput. Sci.*, 89(4) :543–560, 2003.

[21] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Aspartix : Implementing argumentation frameworks using answer-set programming. In *Proceedings of the 24th International Conference on Logic Programming, ICLP 2008*, 2008.

[22] Éric Grégoire, Jean-Marie Lagniez, and Bertrand Mazure. An experimentally efficient method for (MSS, CoMSS) partitioning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 2666–2673, 2014.

[23] Jean-Marie Lagniez and Armin Biere. Factoring out assumptions to speed up MUS extraction. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing - SAT 2013*, pages 276–292, 2013.

[24] João Leite and João Martins. Social abstract argumentation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*, pages 2287–2292, 2011.

[25] Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, 40(1) :1–33, 2008.

[26] João Marques-Silva, Federico Heras, Mikolás Janota, Alessandro Previti, and Anton Belov. On computing minimal correction subsets. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, 2013.

[27] Bart Verheij. Two approaches to dialectical argumentation : Admissible sets and argumentation stages. In *Proceedings of the biannual International Conference on Formal and Applied Practical Reasoning, FAPR 1996*, pages 357–368. Universiteit, 1996.

SwarmSAT: un solveur SAT massivement parallèle

Jean-Marie Lagniez **Nicolas Szczepanski** **Sébastien Tabary**
Univ. Lille-Nord de France, CRIL/CNRS UMR8188, Lens, F-62307 CRIL-CNRS

{lagniez,szczepanski,tabary}@cril.fr

Résumé

Dans ce papier, nous discutons de l'efficacité de la résolution du problème SAT dans un cadre massivement parallèle. Plus précisément, nous proposons une première version d'un solveur SAT, nommé SwarmSAT, fondé sur une approche de type collaborative à la CubeAndConquer et où la partie communication est gérée grâce à la bibliothèque Open MPI. Nous montrons expérimentalement que l'approche de base, qui consiste à diviser l'espace de recherche sous forme de cubes et de les résoudre en parallèle, n'est pas viable en pratique. Ces explications mettront en évidence les axes d'améliorations à apporter à SwarmSAT pour de futurs travaux.

1 Introduction

Longtemps étudié dans un contexte séquentiel pour ses nombreuses applications industrielles (planification, vérification formelle, ...), le problème SAT a fait éclore des algorithmes efficaces et puissants qui s'expliquent à la fois par l'amélioration des solveurs SAT modernes (apprentissage, VSIDS, *watched literal*, ...) et l'accroissement de la puissance de calcul. Aujourd'hui, les solveurs SAT permettent la résolution de problèmes de taille importante contenant des millions de variables et de clauses. Néanmoins, certaines instances restent encore inabordables en raison, en autres, de leur structure. La parallélisation des algorithmes de résolution pour SAT constitue un axe d'amélioration. Ceci est renforcé par le fait qu'à l'heure actuelle la puissance de calcul d'un ordinateur ne se traduit plus par une augmentation des fréquences du microprocesseur, mais par l'augmentation du nombre de cœurs de calcul au sein de celui-ci. Dans ce contexte, nous distinguons deux modèles pour la résolution parallèle du problème SAT : le modèle concurrentiel qui exécute en parallèle plusieurs solveurs séquentiels en concurrence sur le même problème (`Plingeling` [6], `PeneLoPe` [1], ...) ; et le modèle collaboratif (coopératif) fondé sur le paradigme « diviser pour mieux régner » (`CubeAndConquer` [11], `Dolius` [4], ...).

Dans ce papier, nous présentons un solveur modulaire collaboratif, nommé `SwarmSAT`, basé sur une résolution massivement parallèle de cubes générés selon la méthode proposée dans `CubeAndConquer` [11]. Les communications entre les différentes unités de calcul sont gérées grâce à la bibliothèque de communication Open MPI. Dans la partie expérimentation, nous étudions l'impact des différents composants de `SwarmSAT` (qualité des cubes, coût des communications, ...). Puis, nous comparons `SwarmSAT` avec un solveur collaboratif différent nommé `Dolius` [4] et nous discutons des axes d'améliorations possibles.

2 Préliminaire

Nous supposons le lecteur habitué avec les concepts de variables, de clauses et d'interprétations usités dans le contexte de SAT (voir [7], pour plus d'informations). Dans cette section nous nous focalisons uniquement sur la description du solveur collaboratif `CubeAndConquer`[8].

Le solveur `CubeAndConquer`, initialement proposé pour être exécuté sur une grille de calcul, consiste à diviser l'espace de recherche en millions de cubes (un cube (ou terme) est une conjonction de littéraux) et à les résoudre en parallèle. Étant donné l'importance des cubes, les auteurs proposent d'utiliser un solveur *look-ahead* de type DPLL pour la génération de ces derniers. Cette approche développe un arbre de recherche dans lequel chaque branche correspond à un cube. La profondeur de cet arbre est bornée par une heuristique. Une fois la génération des cubes terminée, ces derniers sont résolus par un ensemble de solveurs `Plingeling` (de type CDCL) en parallèle.

3 SwarmSAT

Dans cette section, nous décrivons les trois types de processus utilisés dans le solveur `SwarmSAT`, ainsi que la manière dont les communications entre ces derniers sont réalisées.

3.1 Les différents processus

Le scout : Il se charge de la construction des cubes de la même manière que `CubeAndConquer` (ie. utilisation du solveur `march_cc` [12]).

Le worker : C'est un solveur SAT qui se charge de la résolution des cubes. Afin de profiter du travail déjà réalisé et d'obtenir une raison lors de l'échec de la résolution de ce dernier, le cube sera passé en *assumption* et le problème résolu de manière incrémentale (voir [3] pour plus d'informations). Ainsi, le résultat retourné par le solveur peut être de deux types : soit il répond SAT et le problème est donc montré globalement satisfaisable ; soit le cube est réfuté. Dans ce dernier cas, il est possible, grâce à la notion d'*assumption*, d'extraire une raison de l'insatisfaisabilité du cube. Si la clause ainsi générée est vide, l'insatisfaisabilité est donc montré globalement, sinon l'insatisfaisabilité est locale et le *worker* demande un nouveau cube à résoudre.

Le master : Il se charge de l'initialisation et de la communication entre les *workers* et le *scout*. C'est ce dernier qui va s'occuper de gérer les cubes produits par le *scout* (ils sont stockés dans une file) et qui les distribue au *worker* en attente. Le *master* a aussi la tâche de centraliser les clauses provenant des cubes déjà prouvés insatisfaisables. Grâce à ces dernières, il a également la possibilité de ne pas considérer un cube qui contredit une des clauses.

3.2 Communication dans `SwarmSAT`

Les communications sont réalisées point-à-point via des messages non-bloquants. Elles sont utilisées pour la transmission des cubes ainsi que le passage de certaines clauses des *workers* au *master* ou encore pour la transmission de la solution. Remarquons que dans la version actuelle de `SwarmSAT`, aucune des clauses apprises n'est échangée entre les *workers*.

4 Mode de fonctionnement

`SwarmSAT` est un solveur modulaire qui permet d'ajouter facilement de nouveaux types de *workers*, de gérer une création de cubes dynamique ou statique et aussi de résoudre une instance du problème SAT en parallèle que ce soit en mode collaboratif (à la `CubeAndConquer`) ou concurrentiel (à la `ppfolio` [13]).

La gestion de l'ajout d'un nouveau solveur est ainsi réalisée grâce à l' implémentation d'une interface écrite en C++. Elle permet de définir les méthodes nécessaires à la gestion des communications via Open MPI. Dans la version expérimentée de ce papier, nous avons considéré les solveurs `Glucose` [5] et `MiniSat` [10] pour implémenter les *workers*.

En ce qui concerne le scout, nous utilisons la version de `march_cc` fournie par les auteurs de `CubeAndConquer`. Cependant, nous avons aussi proposé une version dynamique qui, contrairement à la version initiale, permet de rendre disponibles les cubes durant leur génération (dans `CubeAndConquer` il fallait parfois attendre plus de 20 minutes pour obtenir tous les cubes). Cela nous permet d'anticiper la résolution des cubes même si en contrepartie, certains cubes auraient été prouvés insatisfaisables durant la phase de création.

Dans la suite, nous nommerons $\texttt{SwarmSAT}(\texttt{W}, \texttt{S}, \texttt{G})_{\texttt{C}}$ pour signifier que nous utilisons comme *worker* des solveurs de type `W` (`Glucose`, `MiniSat`, ...) et un *scout* utilisant la méthode `S` pour la génération des cubes. La lettre `G` peut prendre comme valeur `dynamic` ou `static` en fonction de la disponibilité des cubes (directement ou après leur création) tandis que `C` permet de connaître le nombre de workers utilisés.

5 Expérimentations

L'ensemble des expérimentations ont été réalisées sur 64 cœurs grâce à deux nœuds de 32 cœurs. Les nœuds de calcul sont des Dell R910 avec 4 Intel Xeon X7550 contenant chacun huit cœurs. Chaque unité de calcul dispose d'un contrôleur Gigabit Ethernet et de 256 Go de RAM.

Les expérimentations présentées dans ce qui suit utilisent le même protocole (instances et temps de résolution) que celui proposé par les auteurs de `Dolius` [4]. Plus précisément, nous considérons 20 minutes comme limite de temps réel. Les instances sélectionnées sont issues de la compétition SAT 2013 (*application track*) [2]. Une instance est sélectionnée si elle a été résolue par au moins un des cinq premiers solveurs parallèles de la compétition et ne peut être résolue par tous ces solveurs. Cela permet d'avoir des instances difficiles et diversifiées : 18 satisfaisables et 33 insatisfaisables.

Dans `SwarmSAT`, étant donné la nature du *scout* (durée limitée) et du *master* (peu de travail), nous avons choisi de faire cohabiter ces derniers avec les *workers*.

Nous comparons le solveur `Dolius` [4] avec différentes versions de `SwarmSAT` présentées subséquemment :

- S(G,∅,∅)$_{64}$ pour SwarmSAT(Glucose$_{rdm}$, ∅, ∅)$_{64}$: la génération des cubes étant vide, cette version est un solveur concurrentiel. Afin que les solveurs rentrent en concurrence nous utilisons une version de Glucose avec une graine aléatoire différente pour chaque *worker* ;
- S(G,m,d)$_{64}$ pour SwarmSAT(Glucose, march_cc, dynamic)$_{64}$: nous utilisons 64 *workers* de type Glucose et march_cc en mode dynamique pour la génération des cubes ;
- S(M,m,d)$_{64}$ pour SwarmSAT(MiniSat, march_cc, dynamic)$_{64}$: nous utilisons 64 *workers* de type MiniSat et march_cc en mode dynamique pour la génération des cubes ;
- S(G,m,s)$_{64}$ pour SwarmSAT(Glucose, march_cc, static)$_{64}$: nous utilisons comme *workers* 64 solveurs Glucose et march_cc en mode statique. Ici, les *workers* tentent de résoudre l'instance en deux étapes. Dans un premier temps, sans cube durant le temps nécessaire à l'exécution du *scout*. Puis dans un seconds temps, via les cubes.

Solveur	SAT(18)	UNSAT (33)	Total (51)
S(G,∅,∅)$_{64}$	12	12	24
S(G,m,s)$_{64}$	10	11	21
S(M,m,d)$_{64}$	10	1	11
S(G,m,d)$_{64}$	6	4	10
Dolius	**13**	**24**	**37**

TABLE 1 – Résultats des différentes versions de SwarmSAT et de Dolius

Le tableau 1 reporte les résultats obtenus par les différentes approches expérimentées dans ce papier. Tout d'abord, nous discutons des résultats obtenus par le solveur Dolius. Nous pouvons remarquer que ce dernier est significativement meilleur sur les instances insatisfaisables (24 instances insatisfaisables résolues) que les différentes versions de SwarmSAT testées (12 pour la meilleure version de SwarmSAT). Cela s'explique largement par le fait que contrairement à SwarmSAT, Dolius permet d'échanger les clauses apprises entre les différentes unités de calcul. Cet échange d'informations permet de guider les solveurs plus rapidement vers la génération d'une clause vide. Pour ce qui est des instances satisfaisables, nous pouvons voir que l'impact de l'échange d'informations n'est pas aussi prépondérant et qu'une version concurrentielle simple telle que S(G,∅,∅)$_{64}$ permet de résoudre approximativement le même nombre d'instances (13 pour Dolius et 12 pour S(G,∅,∅)$_{64}$).

Concernant les résultats obtenus par les différentes versions de SwarmSAT, nous observons qu'une division de l'espace de recherche à l'aide de cubes, telle qu'elle a été proposée dans [11], n'est pas efficace si elle n'est pas accompagnée d'un échange d'informations

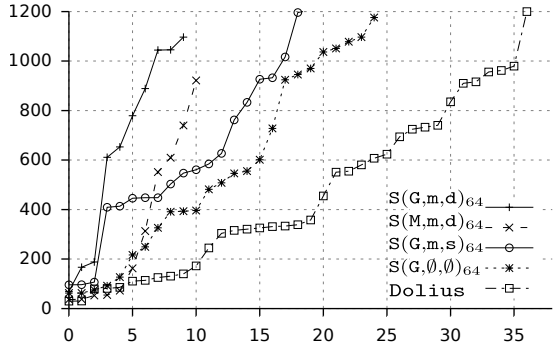


FIGURE 1 – Nombre d'instances résolues en fonction du temps des SwarmSATs et Dolius.

entre les solveurs. En effet, la version concurrentielle de SwarmSAT S(G,∅,∅)$_{64}$, qui ne divise pas l'espace de recherche à l'aide de cubes, fournie les meilleurs résultats (12 instances satisfaisables résolues et 12 instances insatisfaisables résolues). De plus, comme le montre la figure 1, S(G,∅,∅)$_{64}$ résout aussi plus rapidement les instances. Afin de vérifier que la perte de performances de l'approche basée sur les cubes n'était pas due au coût de communication, nous avons mesuré ces derniers. Il s'avère que le temps de transfert des cubes du *master* aux *workers* ainsi que le passage des clauses apprises des *workers* au *master* est insignifiant (moins de 2 secondes au total).

Les expérimentations ne permettent pas de conclure sur la qualité des cubes. En effet, bien que S(G,m,s)$_{64}$ résolve significativement plus d'instances (10 instances satisfaisables et 11 instances insatisfaisables) que S(G,m,d)$_{64}$ (6 instances satisfaisables et 4 instances insatisfaisables) et S(M,m,d)$_{64}$ (10 instances satisfiables et 1 instance insatisfaisable), dans la plupart des cas cela est une conséquence du fait que la génération de tous les cubes est trop coûteuse en temps. Pour 33 instances la génération ne termine pas et pour les 18 instances restantes, cette dernière nécessite en moyenne 296.27 secondes du temps alloué à la résolution. Ces chiffres montrent que dans la majeure partie du temps S(G,m,s)$_{64}$ est équivalent à S(G,∅,∅)$_{64}$. Cela explique largement les performances de S(G,m,s)$_{64}$.

Finalement, nous pouvons voir que le choix du solveur utilisé pour implémenter les *workers* est prépondérant. Ainsi, le choix du solveur MiniSat permet de résoudre plus facilement les instances satisfaisables tandis qu'utiliser Glucose permet d'être plus efficace sur les instances insatisfaisables. Ces résultats s'expliquent par le fait que cette différence de performance en fonction de la satisfaisabilité de l'instance existe déjà entre ces deux solveurs lorsqu'ils sont exécutés séquentiellement. Néanmoins, il semble que dans le cas d'une résolution incrémentale le phénomène est accentué.

6 Conclusion et perspectives

Ce papier présente un nouveau solveur SAT, nommé `SwarmSAT`, largement inspiré de `CubeAndConquer` et dédié au massivement parallèle. Cette première mouture utilise pour la génération des cubes le solveur `march_cc` [11] et comme *workers* l'un des deux solveurs de l'état de l'art `Glucose` et `MiniSat`. Les résultats expérimentaux montrent que dans le cadre de la résolution d'instances industrielles `SwarmSAT` (plus généralement, les approches fondées sur `CubeAndConquer`) ne sont pas efficaces en pratique (sauf dans le cas où les instances seraient de type académique).

Bien que les résultats obtenus dans ce papier ne soient pas à la hauteur de nos attentes, les travaux réalisés autour de la création du solveur `SwarmSAT` forment une base solide pour de nombreuses extensions. En effet, nous avons la possibilité d'actionner plusieurs leviers afin d'améliorer notre approche.

En ce qui concerne la génération des cubes, de nombreuses pistes restent à explorer. Dans un premier temps, nous envisageons de modifier le solveur `march_cc` afin de contrôler le temps nécessaire à la création des cubes. Par exemple, nous pouvons réduire la profondeur de l'arbre de recherche exploré. Nous examinons aussi la possibilité d'itérer le processus de génération de cubes. Plus précisément, nous pourrions dans un premier temps générer des cubes avec une petite profondeur d'arbre afin de fournir rapidement des cubes aux *workers* et de les « améliorer » en augmentant cette dernière. Il serait aussi possible de régénérer un nouveau pool de cubes si ces derniers sont considérés mauvais par les *workers* (par exemple si les solveurs n'avancent pas dans la résolution des cubes).

Un fait marquant des expérimentations est que pour beaucoup d'instances le nombre de cubes résolus est insignifiant (par exemple pour `bob12s06` seulement 4% des cubes ont été traités). Une telle situation peut conduire à focaliser la recherche sur une seule partie de l'espace de recherche. Cela peut être très préjudiciable lorsque nous observons l'impact des redémarrages dans les solveurs SAT modernes. Afin d'atténuer ce phénomène, nous envisageons de faire en sorte que des cubes puissent obtenir le statut indéterminé. De cette manière un *worker* peut libérer un cube et en choisir un autre. Par la suite, le cube ainsi libéré pourra être retravaillé.

Une autre piste d'amélioration concerne l'échange d'informations entre les *workers*. Notons que cela est d'une certaine manière déjà réalisée via le transfert des clauses apprises lorsqu'un cube est réfuté. Il est tout à fait envisageable de partager des clauses entre les différentes unités de calcul. Cependant, souhaitant rendre applicable `SwarmSAT` dans le cadre massivement parallèle, une attention particulière devra être apportée à la manière donc cet échange sera réalisé. Pour cela, nous pourrons nous inspirer des travaux présentés dans [1] (pour le choix des clauses à partager) et [9] (pour réaliser cet échange).

Finalement, au vu des résultats obtenus par $\texttt{S(G,m,d)}_{64}$ et $\texttt{S(M,m,d)}_{64}$, il semblerait qu'il y ait fort à gagner à utiliser plusieurs types de solveurs pour implémenter les *workers*.

Références

[1] Gilles A., B. Hoessen, S. Jabbour, J-M. Lagniez, and C. Piette. PeneLoPe, a Parallel Clause-Freezer Solver. In *SAT'12*, pages 43–44, 2012.

[2] A. Belov A. Balint, M. Heule and M. Jarvisalo. The application and the hard combinatorial benchmarks in sat competition 2013. In *SAT'13*, page 99, 2013.

[3] G. Audemard, A. Biere, J-M. Lagniez, and L. Simon. Améliorer SAT dans le cadre incrémental. *RIA*, pages 593–614, 2014.

[4] G. Audemard, B. Hoessen, S. Jabbour, and C. Piette. Dolius : A distributed parallel sat solving framework. In *POS'14*, 2014.

[5] G. Audemard and L. Simon. Predicting learnt clauses quality in modern sat solvers. In *IJCAI'09*, pages 399–404, 2009.

[6] A. Biere. Lingeling, plingeling and treengeling entering the sat competition 2013. In *SAT'13 : Solver and Benchmarks Descriptions*, page 51, 2013.

[7] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, 2009.

[8] C. Biró, G. Kovásznai, A. Biere, G. Kusper, and G. Geda. Cube-and-conquer approach for sat solving on grids. *Annales Mathematicae et Informaticae*, pages 9–21, 2013.

[9] T. Ehlers, D. Nowotka, and P. Sieweck. Communication in massively-parallel sat solving. In *ICTAI'14*, pages 709–716, 2014.

[10] N. Eén and N. Sörensson. An extensible sat-solver. In *SAT'03*, pages 502–518, 2003.

[11] M. Heule, O. Kullmann, S. Wieringa, and A. Biere. Cube and conquer guiding CDCL SAT solvers by lookaheads. In *HVC'11*, pages 50–65, 2011.

[12] Marijn Heule and Hans van Maaren. March_dl : Adding adaptive heuristics and a new branching strategy. *JSAT*, (1-4) :47–59, 2006.

[13] Olivier Roussel. Description of ppfolio. 2011.

Trois hyper-heuristiques pour le problème d'affectation de fréquence dans un réseau cellulaire

Yasmine Lahsinat[1] **Belaid Benhamou**[2] **Dalila Boughaci**[1]

[1] LRIA-FEI/USTHB Alger 16111, Algérie
[2] Université Aix-Marseille, LSIS
Domaine universitaire de Saint Jérôme, Avenue Escadrille Normandie Niemen
13397 MARSEILLE Cedex 20
{ylahsinat, dboughaci}@usthb.dz belaid.benhamou@univ-amu.fr

Résumé

Dans cet article, nous nous intéressons aux approches hyper-heuristiques pour résoudre le problème de Minimisation d'Interférences lors de l'Affectation de Fréquence noté (MI-FAP) dans un réseau cellulaire. Nous proposons trois variantes d'hyper-heuristiques. Ces hyper-heuristiques se distinguent entres elles de par la stratégie utilisée pour sélectionner l'heuristique de bas niveau à appliquer. La première se base sur une fonction de choix (HFC) pour sélectionner l'heuristique la plus appropriée. La seconde choisit aléatoirement l'heuristique de bas niveau et est notée (HHA). La troisième, nommée stochastique hyper-heuristique (SHH) combine les deux stratégies de sélection : la fonction de choix et la stratégie aléatoire pour sélectionner l'heuristique de bas niveau. La sélection de l'heuristique se fait selon l'une des deux stratégies en utilisant une probabilité fixe. Nous proposons aussi une adaptation d'une métaheuristique récente pour résoudre le problème MI-FAP. A cet effet, nous avons retenu l'algorithme d'Optimisation Basée sur la Biogéographie noté (OBB). Des expérimentations sont réalisées sur des benchmarks publics de diverses tailles avec pour objectif de prouver l'efficacité de nos approches hyper-heuristiques. Les résultats trouvés sont comparés à ceux obtenus par la méthode d'Optimisation basée sur la Biogéographie (OBB). Les résultats obtenus par les approches hyper-heuristiques sont nettement meilleures que ceux obtenus par la métaheuristique(OBB).Les résultats expérimentaux montrent l'efficacité des deux méthodes SHH et HHA comparativement à la méthode HFC pour résoudre le MI-FAP.

Abstract

In this paper, we are interested in hyper-heuristic approaches for solving the Minimum Interference Frequency Assignment Problem (MI-FAP) in cellular network. We propose three variants of hyper-heuristics. The Hyper-heuristics vary from each other in the strategy used to select the heuristics. First, we propose a Choice Function based Hyper-Heuristic (CFH) which uses a function to select the appropriate heuristic. The second approach is a Random Hyper-Heuristic (RHH) that selects randomly the heuristic to apply. The third one is a Stochastic Hyper-Heuristic (SHH). SHH combines both the choice function and the randomness strategies. SHH uses a walk probability to decide, among the selection strategies, the heuristic that is going to be applied. All of the algorithms have been tested, evaluated on public available benchmarks and compared with the Biogeography Based Optimization (BBO) metaheuristic. The experiments show the effectiveness of the proposed Hyper-Heuristics methods compared to BBO. On other hand, the experimental results prove that the performances of both SHH and RHH methods are better than the CFH method in solving the MI-FAP.

1 Introduction

Un réseau radio mobile passe plusieurs phases lors de son design telles que le placement des antennes et l'affectation de fréquences. Cette dernière est une étape cruciale dans le design d'un réseau de télécommunication. C'est un problème d'optimisation combinatoire bien connu qui revient à trouver un plan de fréquences optimal de façon à satisfaire la demande en trafic et à assurer la qualité de service tout en minimisant les interférences co-canal et canaux adjacents. Le problème de minimisation des interférences lors de l'affectation de fréquence en anglais *Minimum Interference Frequency Assignment Problem* et noté (MI-FAP) a été introduit dans les années soixante par Met-

zeger [21], il a depuis suscité l'intérêt de beaucoup de chercheurs [1, 11, 17].

En pratique les opérateurs de téléphonies mobile disposent d'un nombre limité de fréquences radio représentatnt une ressource rare et couteuse de l'opérateur. Ils disposent aussi d'un nombre d'émetteurs / récepteurs ($TRXs$) auxquels il faut affecter des fréquences pour assurer le service. Afin de gérer ces deux faits, les opérateurs font appel au principe de réutilisation de fréquences. Ce dernier n'a pas que des avantages, il a pour effet d'engendrer des interférences particulièrement lorsque des ($TRXs$) proches géographiquement utilisent la même fréquence ou des fréquences adjacentes. En théorie le problème d'affectation de fréquences est un problème NP-Difficile [13].

D'un point de vue mathématique le problème MI-FAP est une généralisation du problème de coloration de graphe. Le problème de minimisation d'interférences lors de l'affectation des fréquences est un problème d'optimisation combinatoire dont l'issue est la découverte d'une affectation de fréquences pour chaque $TRXs$ du réseau avec le minimum possible d'interférences et le respect des contraintes ci-dessous nommées :

1. Les contraintes de cellules :
   - Les contraintes de séparation : les émetteurs / récepteurs d'une même cellule ne doivent pas utiliser la même fréquence.
   - Les canaux bloqués : certaines fréquences ne sont pas opérationnelles dans certaines cellules.
2. Les contraintes des interférences inter-cellules :
   - Les interférences Co-channel : ces dernières sont engendrées dans le cas où deux $TRXs$ proches géographiquement utilisent la même fréquence (f).
   - Les interférences Canal-adjacent : ces dernières sont engendrées dans le cas où deux $TRXs$ proches géographiquement utilisent des fréquences adjacentes(f), (f+1 / f-1).

Diverses méthodes et approches pour résoudre le problème MI-FAP ont été proposées. Nous pouvons trouver des approches évolutionnaires telles que EAs et des algorithmes d'optimisation par colonies de fourmis (ACO / EAs) [18], un algorithme de recherche Tabu [8, 5] et ses améliorations [22]. On peut citer aussi un algorithme de recuit simulé utilisé dans [9], un algorithme Branch and cut proposé dans [12], un algorithme culturel [2], ainsi que des approches de résolution par algorithme hybride utilisant l'algorithme génétique et la recherche Tabu [20] et d'autre approches hybrides que l'on trouve dans [19]. Une résolution par hyper-heuristique parallèles a été proposé dans [24] sur des données non disponibles.

Dans ce papier, nous proposons trois hyper-heuristiques qui se basent sur différents mécanismes lors de la sélection de l'heuristique de bas niveau à appliquer pour résoudre le MI-FAP. La première hyper-heuristique se base sur une fonction de choix, elle est notée(HFC), la deuxième choisit aléatoirement l'heuristique à appliquer, elle est notée (HHA) et enfin la troisième nommée Stochastique Hyper-Heuristique (SHH) combine les deux précédentes méthodes de sélection pour choisir l'heuristique de bas niveau à utiliser. Des expérimentations numériques sont réalisées sur des différents benchmarks dans le but de tester et de prouver l'efficacité des approches proposées. Nous comparons les résultats obtenus par les approches hyper-heuristiques avec ceux obtenus par l'adaptation de l'algorithme d'optimisation basée sur la Biogéographie pour le problème MI-FAP. Cette dernière est une des méta-heuristiques les plus récentes qui a retenu notre attention. Nous étions alors curieux de voir ses performances sur le problème MI-FAP.

Le reste de ce papier est organisé comme suit : la section 2 présente le problème MI-FAP. La section 3 introduit les approches hyper-heuristique proposées. La section 4 présente brièvement l'algorithme OBB, que nous adaptons pour le problème MI-FAP. Quelques résultats expérimentaux sont donnés dans la section 5. Enfin, la section 6 donne une conclusion et quelques perspectives inhérentes à ce travail.

2 Description et modélisation du problème

Le problème de minimisation des interférences lors de l'affectation des fréquences dans un réseau cellulaire peut être énoncé comme suit : considérons un réseau cellulaire N découpé en nc cellules : $C=\{C_1, C_2 \dots C_i, \dots C_{nc}\}$. Le réseau dispose d'un intervalle $[F_{min}, F_{max}]$ de fréquences opérationnelles et $NF = |F_{min} - F_{max}|$ le nombre de fréquences. Pour chaque cellule, on dispose d'un ensemble de fréquences opérationnelles. Les interférences entre les (TRXs) sont représentées par les deux matrices IM_{co} respectivement IM_{adj} définies comme suit :

- La matrice $IM_{co}[nc][nc]$: une matrice de rang $nc \times nc$ où l'élément $IM_{co}[i][j]$ exprime le niveau d'interférence co-channel dans le cas où des émetteurs/ récepteurs de cellules i et j utiliseraient une même fréquence (f).
- La matrice $IM_{adj}[nc][nc]$: une matrice de rang

$nc \times nc$ où l'élément $IM_{adj}[i][j]$ exprime le niveau d'interférence canal-adjacent dans le cas où des émetteurs/ récepteurs de cellules i et j utiliseraient des fréquences adjacentes (f), $(f + 1/ f - 1)$.

Une solution $S = \{f_1, f_2, f_i \dots f_d\}$ du problème MI-FAP est une affectation de fréquences aux d émetteurs / récepteurs du réseau qui satisfait les contraintes des cellules. Chaque élément f_i de S correspond à la fréquence assignée au i^{me} émetteur/ récepteur du réseau.

Le problème d'affectation de fréquences dans un réseau cellulaire consiste à trouver une allocation de fréquence à chaque TRX du réseau qui minimise les interférences. C'est un problème d'optimisation combinatoire dont la fonction objectif permet d'évaluer le niveau d'interférences généré par toute affectation S de fréquences aux différents $(TRXs)$ du réseau. Cette fonction que nous cherchons à minimiser est exprimée par la formule suivante [15] :

$$F(S) = \sum_{i=1}^{nc} \sum_{j=1}^{nc} (Y_{ij} \times IM_{co}[i][j] + Z_{ij} \times IM_{adj}[i][j]) * w_i (1)$$

où :

- w_i : est un poids associé à chaque cellule. Il dépend du trafic de la cellule.
- Y_{ij} : est une variable de décision, qui prend 1 si des $(TRXs)$ de cellules i et j utilisent la même fréquence dans S, 0 sinon.
- $IM_{co}[i][j]$: est un élément de la matrice IM_{co} qui exprime le niveau d'interférence provoqué par l'affectation dans S d'une même fréquence à des $(TRXs)$ des cellules i et j.
- Z_{ij} : est une variable de décision, qui prend 1 si des $(TRXs)$ de cellules i et j utilisent dans S des fréquences adjacentes, 0 sinon.
- $IM_{adj}[i][j]$: est un élément de la matrice IM_{adj} qui exprime le niveau d'interférence provoqué par l'affectation de fréquences adjacentes dans S à des $(TRXs)$ des cellules i et j.

Si Sol représente l'ensemble des solutions, la valeur optimale de la fonction objectif est alors $F^* = MIN_{S \in Sol}(F(S))$ et la solution réalisant cette valeur optimale est notée S^*.

3 Des approches Hyper-heuristiques pour le MI-FAP

Les hyper-heuristiques sont des méthodologies de recherche travaillant à un haut niveau d'abstraction et d'indépendance par rapport au problème d'optimisation traité. Elles sont définies dans [4] comme *"heuristics to choose heuristics"*, c'est-à-dire des heuristiques permettant de choisir d'autres heuristiques.

Les hyper-heuristiques sont des (méta) heuristiques de haut niveau qui manipulent un ensemble d'heuristiques de bas niveau. L'objectif est de gérer la sélection des heuristiques de bas niveau à appliquer parmi un ensemble d'heuristiques. La stratégie de sélection de l'heuristique de bas niveau est très importante pour l'hyper-heuristique. Elle doit avoir l'habilité de choisir l'heuristique la plus appropriée à chaque itération durant le processus de recherche de solution et assurer la bonne collaboration entre les différentes heuristiques manipulées.

Dans cet article, nous proposons trois approches fondées sur l'hyper-heuristique pour résoudre le problème MI-FAP. La principale différence entre ces trois approches se situe dans la stratégie utilisée pour la sélection de l'heuristique de bas niveau. Les approches hyper-heuristiques proposées dans ce travail manipulent cinq heuristiques de bas niveau. Nous présentons dans ce qui suit les cinq heuristiques de bas niveau que nous proposons pour le problème MI-FAP :

L'heuristique *h1* : *h1* implémente une recherche locale de base et simple à mettre en oeuvre. C'est une procédure itérative qui démarre d'une solution initiale générée aléatoirement X. A chaque itération, une solution voisine est obtenue en appliquant un mouvement. Pour le problème MI-FAP, le mouvement consiste à modifier la fréquence d'un (TRX). Le choix du (TRX) se fait aléatoirement parmi l'ensemble des $(TRXs)$. Le choix de la nouvelle fréquence à affecter se fait avec le respect des contraintes de cellules. Le processus est répété un nombre de fois avec pour objectif l'amélioration de la solution.

L'heuristique *h2* : *h2* est une recherche locale stochastique. Elle prend en entrée la meilleure solution trouvée, puis génère un voisin en appliquant un mouvement. Comme pour l'heuristique $h1$ le mouvement consiste à modifier la fréquence d'un (TRX). Pour $h2$ le mouvement s'effectue en utilisant une probabilité fixe $wp > 0$. La probabilité $wp > 0$ intervient lors du choix du (TRX) dont la fréquence sera modifiée. Le choix du (TRX) se fait selon les deux critères suivants :

- Le premier critère consiste à sélectionner un

(TRX) parmi l'ensemble des ($TRXs$) de manière aléatoire avec une probabilité $wp > 0$. La règle du mouvement ici est similaire à celle appliquée dans $h1$.

- Le deuxième critère consiste à choisir, avec une probabilité $1 - wp$, un (TRX) appartenant à la cellule dont l'affectation des fréquences - à ses différents ($TRXs$)- a engendré le plus d'interférences. On remplace la fréquence actuelle du (TRX) par la fréquence la moins assignée aux ($TRXs$) des cellules prochent géographiquement de la cellule trouvée.

L'heuristique *h3* : *h3* reprend le principe d'un opérateur de mutation qui selon un paramètre de mutation noté ($mrate$) appliquera ou non, pour chaque (TRX) d'une solution X, une perturbation. C'est à dire que l'affectation de fréquence actuelle de chaque (TRX) sera modifiée ou non selon le paramètre $mrate$. La solution obtenue est ensuite passée comme paramètre d'entrée (solution de départ) à une recherche locale pour l'améliorer. La recherche locale appliquée est implémentée par l'heuristique (h1).

L'heuristique *h4* : *h4* s'inspire de l'opérateur de croisement d'un algorithme génétique afin de créer une nouvelle solution. Pour la mettre en oeuvre nous proposons de combiner la meilleure solution avec la moins bonne solution obtenues à l'appel de cette heuristique. Ensuite on applique un opérateur de mutation sur la nouvelle solution obtenue. L'opérateur de mutation est celui proposé dans l'heuristique *h3*. Suite à la phase de mutation, nous appliquons la recherche local stochastique implémentée par l'heuristique *h2* afin d'améliorer la solution obtenue.

L'heuristique *h5* : *h5* s'inspire tout comme l'heuristique *h4* de l'opérateur de croisement d'un algorithme génétique dans le but de créer une nouvelle solution. Pour la mettre en oeuvre nous proposons de combiner la meilleure solution avec une nouvelle solution générée aléatoirement. Ensuite on applique l'heuristique *h3* à la solution obtenue. Cette solution est ensuite améliorée en utilisant la recherche locale stochastique imlplémentée par *h2*.

Nous sommes maintenant en mesure de décrire les différentes hyper-heuristiques que nous proposons pour résoudre le problème MI-FAP.

3.1 L'hyper-heuristique fonction de choix (HFC)

Dans un premier temps, nous proposons une approche hyper-heuristique basée sur une fonction de choix pour sélectionner l'heuristique de bas niveau à appeler. Cette fonction de choix est une adaptation au problème MI-FAP de la fonction utilisée dans [6, 7] pour le problème de scheduling. Elle associe à chaque heuristique de bas niveau un score qui correspond à son efficacité. Ce score permettra de sélectionner l'heuristique la plus adéquate à chaque étape du processus de recherche.

Algorithm 1 Hyper-heuristique Fonction Choix (HFC)

Require: Une instance du MI-FAP, un ensemble d'heuristiques de bas niveau, la fonction de choix, les paramètres α, β et γ.
Ensure: meilleure solution S^*.
1: Génerer une solution initiale S;
2: Evaluer la qualité $F(S)$ de la solution S donnée par la fonction F de l'équation (1);
3: $S^*= S$; $F^*= F(S)$;
4: **while** (le critère d'arrêt non vérifié) **do**
5: Pour chaque heuristique h_i, calculer son score en utilisant la fonction de choix $score(h_i)$.
6: Sélection de l'heurisituqe h_i ayant le score le plus élevé.
7: Appliquer h_i à S afin d'obtenir une nouvelle solution $S^{'}$ de qualité $F(S')$.
8: **if** $F(S') < F^*$ **then**
9: $S^*=S'$.
10: $F^*=F(S')$.
11: **end if**
12: **end while**

Afin d'attribuer un score à chaque heuristique de bas niveau, la fonction de choix se base sur les éléments suivants :

1. La performance récente de chaque heuristique (représentée par g_1).
2. La performance récente de paires d'heuristiques (représentée par g_2).
3. La récence du dernier appel de l'heuristique (représentée par g_3).

Ainsi nous avons :
$\forall_i, g_1(h_i) = \sum_n \alpha^{n-1} \frac{I_n(h_i)}{T_n(h_i)}$
$\forall_i, g_2(h_{RL}, h_i) = \sum_n \beta^{n-1} \frac{I_n(h_{LL}, h_i))}{T_n(h_{RL}, h_i))}$
$\forall_i, g_3(h_i) = elapsedTime(h_i)$
$\alpha, \beta \in [0, 1], \gamma \in R$

Avec :

- h_i : la i^{me} heuristique et h_{RL} la dernière heuristique lancée.

- α, β et γ sont des poids reflétant l'importance de chaque terme. Ils ont été fixés empiriquement.
- Les valeurs I_n et T_n donnent l'information sur la qualité de la solution et le temps CPU de la i^{ime} heuristique.

Le score de chaque chaque heuristique de bas niveau est définit comme suit :

$\forall_i, score(h_i) = \alpha g_1(h_i) + \beta g_2(h_{RL}, h_i)) + \gamma g_3(h_i)$

- g_1 est la performance récente de l'heuristique (h_i).
- g_2 est la performance récente de la paire d'heuristiques : l'heuristique (h_i) et celle la précédant (h_{RL}).
- g_3 mesure la récence du dernier appel de l'heuristique (h_i).

Les termes g_1 et g_2 sont les éléments qui permettent d'intensifier la recherche tandis que l'élément g_3 apporte de la diversification et cela en favorisant les heuristiques qui n'ont pas été choisies depuis un long moment.

L'hyper-heuristique basée sur cette fonction de choix est présentée dans l'algorithme 1.

3.2 L'hyper-heuristique aléatoire (HHA)

Algorithm 2 Hyper-Heuristique Aléatoire (HHA).

Require: Une instance du MI-FAP et un ensemble d'heuristiques de bas niveau.
Ensure: meilleure solution S^*.

```
1: génerer une solution initiale S;
2: Evaluer la qualité F(S) de la solution S donnée par la fonction objective de l'équation (1);
3: S*= S ; F*= F(S);
4: while ( le critère d'arrêt non rencontré) do
5:    Sélection aléatoire de l'heurisituqe h_i.
6:    Appliquer h_i à S afin d'obtenir une nouvelle solution S' de qualité F(S').
7:    if F(S) < F* then
8:       S*=S'.
9:       F*=F(S').
10:   end if
11: end while
```

La seconde hyper-heuristique que nous proposons pour le problème MI-FAP sélectionne l'heuristique à appeler de façon aléatoire, ce qui donne une chance à chaque heuristique d'être appelée. Comme il a été souvent mentionné, la force de l' aléatoire réside dans sa capacité à apporter de la diversité. L'hyper-heuristique utilisant cette sélection aléatoire est illustré dans l'algorithme2.

3.3 L'hyper-heuristique stochastique (SHH)

La troisième hyper-heuristique que nous proposons combine les deux stratégies de sélections introduites précédemment : la sélection basée sur la fonction de choix et la sélection aléatoire de l'heuristique à appeler. L'idée est inspirée de la recherche locale stochastique proposée dans [3] pour résoudre le problème de détermination du gagnant dans une enchère combinatoire. A chaque étape, le processus de recherche sélectionne selon une probabilité fixe $wp_1 > 0$ une des deux méthodes de sélection suivantes :

- Sélectionner aléatoirement l'heuristique à appliquer avec une probabilité fixe $wp_1 >0$.
- Sélectionner l'heuristique en utilisant la fonction de choix avec une probabilité $1 - wp_1$.

L'hyper-heuristique stochastique est illustrée par l'algorithme 3.

Algorithm 3 Stochastic Hyper-heuristique (SHH).

Require: Une instance du MI-FAP, un ensemble d'heuristiques, la probabilité wp_1, la fonction de choix, α, β, γ.
Ensure: meilleure solution S^*

```
1: générer une solution initiale S;
2: Evaluer la qualité F(S) de la solution S donnée par la fonction objective de l'équation (1);
3: S*= S ; F*= F; F, F* qualité des solutions S respec S* obtenues grâce à la formule (1).
4: while ( le critère d'arrêt non rencontré) do
5:    r ← un nombre aléatoire entre 0 et 1;
6:    if (r < wp_1) then
7:       h_i=Une heuristique choisie aléatoirement;
8:    else
9:       h_i=Une heuristique choisie selon la fonction de choix;
10:   end if
11:   Appliquer h_i sur S afin d'obtenir une nouvelle solution S' de qualité F'.
12:   if F' < F* then
13:      S*=S'.
14:      F*=F'.
15:   end if
16: end while
```

4 Adaptation de l'algorithme d'optimisation basée sur la Biogéographie pour le MI-FAP

Pour compléter notre étude nous voulons utiliser une méthode à population pour résoudre le problème MI-

FAP. Notre choix est naturellement porté sur la méthode basée sur la Biogéographie qui représente une des méthodes à population les plus récentes. Nous avons été curieux de tester ces performances sur le problème MI-FAP et voir comment elle se comporte en comparaison par rapport à l'approche hyper-heristique. L'optimisation basée sur la biogéographie (OBB) est une méta-heuristique évolutionnaire proposée par *Dan Simon* en 2008 [25]. La méthode s'inspire de la science de la Biogéographie. Cette dernière étudie la distribution des espèces dans leur environnement naturel. La méthode utilise les résultats obtenus par le modèle mathématique de la Biogéographie introduit en 1960 par Robert MacArthur et Edward Wilson. Ce modèle décrit le processus de migration (émigration et immigration) des espèces d'un habitat (île) vers un autre à la recherche de l'habitat comportant les meilleures conditions de vie. L'optimisation basée sur la biogéographie est une méthode qui manipule une population. Chaque individu de la population correspond à un habitat par analogie à la Biogéographie. La qualité de chaque habitat est donnée par *Habitat Suitability Index* (HSI) qui correspond à la qualité de la solution. Chaque habitat est caractérisé par l'ensemble de ses variables indépendantes appelées en anglais *Suitability Index Variable* (SIV), le nombre d'espèces, le taux d'immigration λ et le taux d'émigration μ. La méthode d'optimisation basée sur la Biogéographie est régie par deux processus : un processus de migration et un processus de mutation. La migration dans OBB est un processus probabiliste où les bonnes solutions partagent l'information avec les mauvaises solutions. Les bonnes solutions dans la méthode d'optimisation basée sur la Biogéographie sont celles ayant un bon HSI, un large nombre d'espèces, un taux d'émigration μ élevé et un faible taux d'immigration λ.

$$\lambda = I(1 - k/n) \quad (2)$$
$$\mu = E(k/n) \quad (3)$$

Avec E, I les taux d'émigration respectivement d'immigration maximales, k le nombre d'espèces présentes sur l'île et n le nombre d'espèces maximal que peut contenir une île.

Le processus de migration est un processus probabiliste où les taux d'immigration λ et d'émigration μ de chaque habitat sont utilisés afin de transmettre les caractéristiques entre les habitats. Durant ce processus les bonnes solutions partagent leurs caractéristiques avec les mauvaises solutions. Ainsi les mauvaises solutions sont améliorées en héritant des caractéristiques des bonnes solutions. Nous proposons dans la figure 1 un schéma qui illustre le processus de migration pour le problème MI-FAP. Considérons les deux solutions

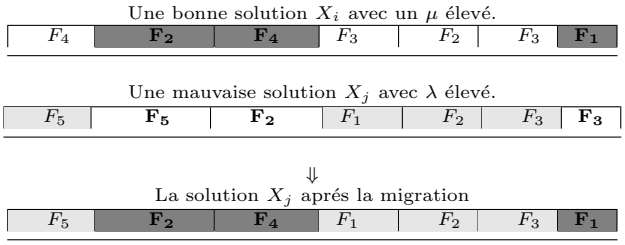


Figure 1: **Schéma illustrant le processus de migration pour le MI-FAP**

X_i et X_j qui représentent des plans de fréquences. Tel que X_i est une bonne solution et X_j une mauvaise solution. A partir de la figure 1, nous pouvons observer qu'au cours du processus de migration, les fréquences assignées aux deuxième, troisième et sixième $TRXs$ dans la solution X_j ont été modifiées. Les nouvelles valeurs de ces fréquences sont celles héritées de la bonne solution X_i.

Nous détaillons ci-après l'adaptation de la méthode (OBB) que nous proposons pour le problème MI-FAP. La population initiale est générée aléatoirement. Chaque habitat de la population correspond à un plan de fréquence. Nous définissons $PopSize$ comme le nombre d'individus de la population. Ensuite, l'algorithme applique à chaque fois les étapes suivantes : évaluer la qualité de chaque solution en calculant son HSI, pour chaque solution mapper le nombre d'espèces à partir du HSI, calculer le taux d'émigration μ et le taux d'immigration λ. Suite à ces opérations, certaines solutions sont choisies pour le processus de migration. Pour rappel cette étape dépend des taux μ et λ. L'algorithme se poursuit avec le processus de mutation. La mutation est un processus qui permet d'apporter de la diversité et éviter la convergence rapide de l'algorithme, il est régi par un paramètre de mutation fixe noté (P_m). Nous proposons d'appliquer la mutation sur la partie la moins bonne de la population. Afin de conserver la meilleure solution dans la population manipulée par (OBB), une stratégie d'élitisme est implémentée. A chaque génération (G), la stratégie d'élitisme que nous adoptons permettra de conserver (n_{elit}) solutions de la génération précédente ($G-1$) si aucun autre individu n'est meilleur qu'eux.

L'algorithme 4 présente le principe de fonctionnement de la méthode OBB pour le problème MI-FAP.

Algorithm 4 La méthode d'optimisation basée sur la Biogéographie pour le MI-FAP

Require: Une instance du MI-FAP, HSI
Ensure: La meilleure solution trouvée S
1: Génération aléatoire de la population initiale
2: **while** (le critère d'arrêt non rencontré) **do**
3: Evaluer la qualité de chaque solution.
4: Calculer le nombre d'espèces.
5: Calculer les taux μ et λ.
6: Appliquer la migration.
7: Appliquer la mutation.
8: **end while**

5 Résultats expérimentaux

Dans le but de valider nos approches hyper-heuristiques et la méthode d'optimisation basée sur la Biogéographie, des tests ont été effectués sur différentes instances du problèmes MI-FAP. Dans nos expériences, nous avons considéré le benchmark du projet de Coopération européenne dans le domaine de la recherche Scientifique et Technique Action 259 (COST 259). Nous avons travaillé sur les données produites par Siemens AG. Les données sont disponibles dans [10]. Nos algorithmes ont été implémentés en utilisant le langage java sous window 7 et sous la machine intel core i5 4GB de RAM. Nous donnons ci-dessous quelques caractéristiques des instances testées.

- Siemens 01 : 506 cellules, 930 $TRXs$, 74 fréquences, 20417 interférences co-channel , 10344 interférences canal adjacent.
- Siemens 02 : 254 cellules, 977 $TRXs$, 82 fréquences, 30982 interférences co-channel, 13970 interférences canal adjacent.
- Siemens 03 : 894 cellules, 1623 $TRXs$, 55 fréquences, 63893 interférences co-channel, 25105 interférences canal adjacent.
- Siemens 04 : 760 cellules, 2785 $TRXs$, 38 fréquences, 113658 interférences co-channel, 64341 interférences canal adjacent.

Les paramètres utilisés dans notre étude expérimentale ont été obtenus après une étude empirique, leurs valeurs sont donnés ci-dessous :

- Les paramètres de l'algorithme de OBB : $PopSize$ 150, n_{elit}=2, P_m=0.1.
- Les paramètres de l'hyper-heuristique basée sur la fonction de choix : α= 0.9, β= 0.1, et γ=1.5
- Paramètre de l'hyper-heuristique stochastique : $wp_1 = 0.27$.
- Le nombre d'itération pour l'heuristique *h1* : $Number_{Iter}$= 50.
- Le nombre d'itération et la probabilité utilisée dans l'heuristique *h2* : Nb_{Iter}=50, wp=0.25.
- Le paramètre de mutation dans *h3* : $mrate$=0.05.
- Le nombre d'itération des recherches locale appliquées dans les heuristiques *h3*, *h4*, *h5* est fixé à : 100.

5.1 Les résultats obtenus

Du fait de la nature non déterministe des algorithmes que nous proposons, des séries d'expériences de 10 tests ont été effectuées pour les différents benchmarks. Le critère d'arrêt des algorithmes est défini en fonction du temps CPU. Ce paramètre est fixé à 1800 secondes. Les résultats présentés dans les tableaux 1 à 4 correspondent aux valeurs moyennes (Mean), la meilleure valeur (Best), la valeur la moins bonne (worst) de la fonction objectif, obtenus sur 10 exécutions pour chacun des algorithmes testés. La valeur (std) correspond à l'écart type entre les différentes exécutions. Les résultats en gras indiquent les meilleurs résultats obtenus en termes de meilleure valeur (Best) et valeur moyenne (Mean). Les résultats présentés dans les tableaux 1 à 4 montrent clairement que les méthodes basée sur l'Hyper-heuristique donnent de meilleurs résultats que OBB.

Statistical Features	**HFC** Cost	**HHA** Cost	**SHH** Cost	**OBB** Cost
Best	47.70	**21.27**	42.79	223.62
Mean	88.78	**41.60**	49.04	250.18
Worst	253.54	98.96	58.95	270.13
Std	60.52	22.93	5.26	13.22

Table 1: Les résultats obtenus sur Siemens 01

Le tableau 1 présente les résultats obtenus par les différentes méthodes sur l'instance Siemens 01. Une colonne est réservée pour présenter les résultats de chaque méthode. Nous pouvons voir que les trois hyper-heuristiques HFC, HHA et SHH surpassent largement la méthode OBB sur cette instance en terme de meilleure (best), moyenne (Mean) et mauvaise solutions trouvées. Nous pouvons constater aussi que l'hyper-heuristique aléatoire (HHA) et l'hyper-heuristique stochastique (SHH) donnent de meilleurs résultats en terme de meilleur, moyenne et moins bonne solutions que l'hyper-heuristique basée sur une fonction de choix. Les résultats entre les deux méthodes HHA et SHH sont comparables en termes de valeur moyenne, HHA est meilleur en terme

de meilleur valeur mais moins bonne en terme moins bonne valeur.

Statistical Features	**HFC** Cost	**HHA** Cost	**SHH** Cost	**OBB** Cost
Best	4990,70	5187.72	**4199.66**	5941,29
Mean	5815,79	5422.59	**4672.32**	6426,40
Worst	6180.81	5672.4	4941.44	7068,98
Std	376.09	137.31	237.64	372.10

Table 2: Les résultats obtenus sur Siemens 02

Le tableau 2 présente les résultats obtenus sur l'instance Siemens 02. On peut voir que les résultats obtenus par l'hyper-heuristique stochastique (SHH) sont meilleurs que ceux de toutes les autre méthodes. Les deux hyper-heuristLes HHA et HFC sont légèrement meilleurs que la méthode OBB. Les résultats obtenus par HHA et HFC sont comparables voir HFC est légèrement meilleure. En effet, HFC est meilleure que HHA en terme de meilleure solution et HHA est légèrement meilleure que HFC en terme de solution moyenne.

Statistical Features	**HFC** Cost	**HHA** Cost	**SHH** Cost	**OBB** Cost
Best	923,12	884.64	**702.48**	1460,65
Mean	1409,34	1004.09	**787.99**	1544,94
Worst	1816.39	1196.19	944.16	1771,72
Std	268.58	90.42	76.63	94.79

Table 3: Les résultats obtenus sur Siemens 03

Le tableau 3 présente les résultats obtenus par les différentes méthodes sur l'instance Siemens 03. Ici aussi, on constate que l'hyper-heuristique SHH présente des résultats meilleurs que ceux de toutes les autres méthodes. Les résultats de HHA sont légèrement meilleurs que ceux de HFC qui est elle même meilleure que la méthode OBB.

Statistical Features	**HFC** Cost	**HHA** Cost	**SHH** Cost	**BBO** Cost
Best	108232.4	**76968.78**	87723.83	110021,78
Mean	110959.6	**89047.31**	91096.05	111314.60
Worst	114512.9	94984.77	99928.78	112774,69
Std	2516.33	5191.25	3635.05	882.01

Table 4: Les résultats obtenus sur Siemens 04

Le tableau 4 contient les résultats obtenus par les différentes méthodes pour l'instance Siemens 04. On peut voir que les méthodes HHA et SHH surpassent les deux autres méthodes HFC et OBB. Les résultats de HHA et SHH sont comparables avec une légère avance pour la méthode aléatoire. Nous pouvons remarquer aussi que les résultats entre OBB et l'hyper-heuristique basée sur une fonction de choix sont comparables.

Nous remarquons que l'écart type obtenu pour les différents tests est relativement petit.

Les résultats obtenus et résumés dans les quatre tableaux précédents montrent clairement l'efficacité et la pertinence de l'approche hyper-heuristique, comparé à la méthode d'optimisation basée sur la Biogéographie. Ceci peut être expliqué par le fait que la méthode d'optimisation basée sur la Biogéographie converge rapidement et stagne, malgré le processus de mutation qui semble être insuffisant pour permettre à l'algorithme de s'échapper des optimums locaux. Les résultats obtenus par l'hyper-heuristique stochastique et l'hyper-heuristique à sélection aléatoire sont meilleures comparées à ceux obtenus par l'hyper-heuristique basée sur une fonction de choix. En effet, cette dernière souffre du manque de diversité, et stagne souvent dans un optimum local. Quant à l'hyper-heuristique aléatoire de part sa stratégie de sélection aléatoire donne une chance à chaque heuristique d'améliorer la solution, ce qui assure une grande diversité. L'efficacité de l'approche hyper-heuristique stochastique s'explique par le bon équilibre entre diversité et intensification. La sélection aléatoire de l'heuristique apporte l'élément de diversification alors que la sélection selon la fonction de choix permet d'intensifier la recherche. Il est évident que l'hyper-heuristique basée sur une fonction de choix reste sensible aux choix des paramètres utilisés. Nous restons convaincus qu'une étude plus approfondie sur les valeurs et l'influence des paramètres utilisés garantira de bien meilleurs résultats. En conséquence, nous pensons qu'avec de bons réglages sur ces paramètres l'hyper-heuristique stochastique finira par l'emporter largement devant l'hyper-heuristique aléatoire. L'hyper-heuristique stochastique semble plus stable et plus robuste.

Instances	**HFC Cost**	**HHA Cost**	**SHH Cost**	**OBB Cost**	**SA-DP Cost**
Siemens 01	12.559	7,33	12,04	47.019	2.20
Siemens 02	51.148	54,39	45,33	65.093	14.27
Siemens 03	81.502	90,95	69	130.387	4.73
Siemens 04	442.302	347,59	370,22	466.566	77.25

Table 5: Comparaison de nos méthodes avec SA-DP

Le problème MI-FAP a été largement étudié sur différents benchmarks autres que ceux que nous avons testés [14, 16]. Nous présentons dans le tableau 5 une comparaison de nos méthodes avec la méthode du recuit simulé combiné à la programmation dynamique [10]. Pour réaliser cette comparaison, nous avons apporté une légère modification à la fonction objectif présentée dans la formule (1). Nous ne prenons plus en compte le poids w_i associé à chaque cellule. La valeur de la fonction objectif correspond alors à la somme des interférences Co-canal et canal adjacent. Ce qui explique la différence entre les résultats présentés dans

le tableau 5 et les résultats présentés dans les tableaux 1-4. Les différentes valeurs reportées dans le tableau 5 correspondent à la valeur moyenne de la fonction objectif obtenue sur dix exécutions. Nos méthodes sont lancées pour un temps CPU fixé à 1800 secondes. La colonne (SA-DP) correspond aux résultats obtenus par la méthode du recuit simulé combiné à la programmation dynamique. Les résultats de la méthode SA-DP ont intégralement été récupérés de [10]. Cette méthode est celle ayant donnée les meilleurs résultats dans la littérature. Néanmoins, nous tenons à signaler qu'il a été reporté dans [10, 23] que ces bons résultats ont été trouvés par la méthode SA-DP après un temps de calcul très important parfois mesuré en semaines. D'autre résultats, moins bon que ceux obtenus par la méthode (SA-DP), sont disponibles sur [10]. On peut trouver aussi dans [22, 23] des résultats obtenus par méthode tabou. Ces derniers restent moins bons que ceux obtenus par (SA-DP).

6 Conclusion

Le travail présenté dans ce papier avait pour objectif d'une part, d'introduire trois approches basées sur l'approche hyper-heuristique pour résoudre le problème d'affectation de fréquence dans un réseau cellulaire. Nous avons d'autre part adapté l'algorithme d'optimisation basée sur la Biogéographie pour le problème MI-FAP. Nous avons présenté, dans un premier temps trois variantes d'hyper-heuristique : une hyper-heuristique basée sur une fonction de choix, une hyper-heuristique à sélection aléatoire et une hyper-heuristique stochastique. Les hyper-heuristiques proposées manipulent un ensemble d'heuristiques de bas niveaux. La différence entre les hyper-heuristiques développées réside dans le mécanisme de sélection de l'heuristique à appeler. Nous avons implémenté puis testé nos méthodes sur différentes instances du problème d'affectation de fréquences. Nous avons pu observer l'efficacité des approches hyper-heuristiques comparées à OBB. Nous avons ensuite comparé nos méthodes à la méthode (SA-DP). Les résultats expérimentaux montrent que les performances des algorithmes proposés restent faibles par rapport à ceux de la méthode SA-DP obtenu après des semaines de temps de calcul. Les résultats que nous avons présentés dans ce papier sont encore en progrès, une analyse expérimentale plus poussée reste à faire. Les heuristiques de bas niveau, les paramètres des algorithmes proposés peuvent être mieux travaillés. Une comparaison avec d'autres algorithmes est planifiée. Ainsi que des tests sur d'autres instances pour lesquels les temps de calculs sont connus est en cours de réalisation.

References

[1] K.I. Aardal, S.P.M. van Hoesel, A.M.C.A. Koster, , C. Mannino, A. Sassano, "Models and solution techniques for frequency assignment problems", *in Annals of Operations Research 153(1)*, pp : 79-129, 2007.

[2] J. Alami and El Imrani, "Using Cultural Algorithm for the Fixed-Spectrum Frequency Assignment Problem", *in the Journal of Mobile Communication 2 (1)*, pp : 1-9 , 2008.

[3] D. Boughaci, B. Benhamou, H. Drias, "A memetic algorithm for the optimal winner determination problem", *in Soft Comput*, 13, 905-917, 2009.

[4] E. K. Burke and G. Kenda, "Search Methodologies : Introduction tutorials in optimization and decision support techniques.", Springer, 2005.

[5] D. Castelino , S. Hurley , And N. Stephens, "A Tabu Search Algorithm for frequency assignment", *in Annals of Operations Research 63*, pp : 301-319, 1996.

[6] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics : A robust optimisation method applied to nurse scheduling. In Parallel Problem Solving from Nature -PPSN VII : 7th International Conference, volume Volume 2439/2002 of Lecture Notes in Computer Science, pages 851860, Granada, Spain, 2002. Springer Berlin/ Heidelberg.

[7] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics : A tool for rapid prototyping in scheduling and optimisation. In Proceedings of the second workshop on Evolutionary Computation in Combinatorial Optimization (EvoCOP2002), volume Volume 2279/2002 of Lecture Notes in Computer Science . Springer LNCS, 2002. ISSN0302-9743 (Print) 1611-3349 (Online).

[8] R. Dorne, J.K. Hao, "Tabu search for graph coloring, T-colorings and set T-colorings", In S. Voss, S. Martello, I.H. Osman, C. Roucairol (Eds.), Meta-heuristics : Advances and Trends in Local Search Paradigms for Optimization. Chapter 6, pp : 77-92, Kluwer.

[9] M. Duque-Anton, D. Kunz, And B. Ruber, "Channel Assignment for Cellular Radio Using Simulated Annealing", *in the IEEE Transactions on Vehicular Technology* 42,1, 1993.

[10] A. Eisenbltter and A. Koster, COST 259. Wireless Flexible Personalized Communications. Available

online at http : //fap.zib.de/problems/COST 259/.

[11] A. Eisenblatter, M. Grotschel, A.M .C.A. Koster, " Frequency planning and ramifications of coloring", *in Discussions Mathematcae Graph Theory 22 (1)*, pp : 51-88, 2002.

[12] M. Fschetti, C. Lepshy, G. Minerva, G.R. Jacur, AND E. Toto, " Frequency assignment in mobile radio systems using branch-and-cut techniques", *in Tech.rep.*, Dipartimento di matematica e informatica, Universit di Udine, Italy, 1997.

[13] W.K. Hale, "Frequency assignment : Theory and applications", *in Proceedings of the IEEE 68(12)*, pp : 1497-1514, 1980.

[14] J.K. Hao, R. Dorne, P. Galinier, "Tabu search for frequency assignment in mobile radio networks ", Journal of Heuristics 4(1), 47-62, 1998.

[15] A. Kuurne, "Mobile Measurement based Frequency Planning in GSM Network. *PhD Thesis*", Helsinki University of Technology, 2001.

[16] X. Lai, J.K. Hao, "'Path Relingking for the Fixed Spectrum Frequency", *Expert Systems with Applications*, 42(10) : 47554767, 2015.

[17] R. Leese, Hurley, S, "Methods and Algorithms for Radio Channel Assignment", *in Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2002.

[18] F. Luna, C. Blum, E. Alba, A.J. Nebro, "ACO vs EAs for solving a real-world frequency assignment problem in GSM networks". *in the Proceedings of the genetic and Evolutionary Computation Conference (GECCO 2007)*, pp : 94-101, 2007.

[19] F. Luna, C. Estebanez, L. Coromoto, J.M Chaves-Gonzelez, A.J. Nebro, R. Aler, C. Segura, M.A. Vega-Rodreguez, E. Alba, J.M. Valls, Miranda, G, Juan Antonio Gomez Pulido, "Optimization algorithms for large-scale real-world instances of the frequency assignment problem", it in Soft Computing. 15(5),pp : 975-990, 2011.

[20] H. Mabed, A. Caminada, J.K. Hao, "Genetic tabu search for robust fixed channel assignment under dynamic traffic data", *in Coput Optim Appl* 50, pp : 483-506 , 2011.

[21] B.H. Metzger, "Spectrum management technique", *in the 38th National ORSA Meeting, Detroit*, pp : 1497-1514, 1970.

[22] R. Montemanni, JNJ. Moon, D.H. Smith, "An Improved Tabu Search Algorithm for the fixed Spectrum Frequency Assignment Problem", *in the IEEE Transactions on Vehicular Technology,*, vol 52,issue 4, pp : 891- 901, 2003.

[23] R. Montemanni, D.H. Smith, "Heuristic manipulation, tabu search and frequency assignment.", *Computers and Operations Research*, vol 37,issue 3, pp : 543-551, 2010.

[24] C. Segura, G. Miranda, C. Leon, "Parallel hyperheuristics for the frequency assignment problem", *in Memetic Computing* 3, pp : 33-49 , 2011.

[25] D. Simon, "Biogeography Based Optimization", in the IEEE Transactions on Evolutionary Computation,vol 12, pp : 702 - 713, 2008.

Modélisation du Social Golfer en SAT via les contraintes ensemblistes

Frédéric Lardeux[1] **Eric Monfroy**[2]

[1] LERIA - Université d'Angers, Angers, France.
[2] LINA - UMR 6241, Université de Nantes, Nantes, France.
Frederic.Lardeux@univ-angers.fr Eric.Monfroy@univ-nantes.fr

Résumé

Les Problèmes de Satisfaction de Contraintes permettent de modéliser de façon expressive et déclarative les problèmes. De leur côté, les solveurs dédiés au problème de satisfaisabilité (SAT) peuvent gérer de gigantesques instances SAT. Nous présentons donc une technique pour modéliser de façon expressive les problèmes sur les contraintes ensemblistes et pour les encoder automatiquement en SAT. Notre technique est expressive et moins sujette aux erreurs. Nous l'appliquons au probème du Social Golfer en essayant de casser des symmétries du problème.

Abstract

Constraint Satisfaction Problems allow one to expressively model problems. On the other hand, propositional satisfiability problem (SAT) solvers can handle huge SAT instances. We thus present a technique to expressively model set constraint problems and to encode them automatically into SAT instances. Our technique is expressive and less error-prone. We apply it to the Social Golfer Problem and to symmetry breaking of the problem.

1 Introduction

La plupart des problèmes combinatoires peuvent être formulés comme des problèmes de satisfaction de contraintes (CSP) [15]. Un CSP est défini par des variables et des contraintes reliant ces variables. Résoudre un CSP consiste à trouver des affectations des variables qui satisfont les contraintes. Une des plus grandes forces des CSP est l'expressivité : les variables peuvent être de différents types (domaines finis, intervalles, ensembles, ...) et les contraintes également (linéaires ou non, arithmétiques, ensembliste, booléennes, ...). Par ailleurs, le problème de satisfiabilité (SAT) [12] est restreint, en terme d'expressivité, aux variables booléennes et aux formules propositionnelles. Cependant, de nos jours, les solveurs SAT peuvent manipuler des instances SAT gigantesques (plusieurs millions de variables et de clauses). Il est donc intéressant d'encoder les CSP en SAT (e.g., [3, 5]) afin de bénéficier de l'expressivité des CSP et de la puissance de SAT.

Nous nous intéressons ici à l'encodage de contraintes ensemblistes CSP en instances SAT. Dans la communauté de la programmation par contraintes, plusieurs systèmes pour les contraintes ensemblistes ont été réalisés (e.g., bibliothèques [14], intégré à un solveur [1]). Ces travaux ont montré que de nombreux problèmes peuvent être modélisés facilement avec des ensembles.

Coder des problèmes (et des contraintes ensemblistes) directement en SAT est fastidieux (e.g., [16] ou [13]). De plus, optimiser les instances en terme de nombre de variables et clauses conduit rapidement à des modèles très complexes et illisibles dans lesquels se glissent facilement des erreurs. Ainsi, notre approche est basée sur l'encodage automatique des contraintes ensemblistes en instances SAT. Pour ce faire, nous définissons des règles ($\Leftrightarrow_{enc}$) qui encodent les contraintes ensemblistes (telles que l'intersection, l'union, l'appartenance ou le cardinal) dans les clauses et variables SAT correspondantes. L'avantage est que le langage de modélisation (i.e., les contraintes ensemblistes classiques) est déclaratif, expressif, simple et lisible. Nous avons utilisé cette technique pour divers problèmes, et les instances SAT automatiquement générées sont de complexité similaire aux instances générées "à la main". De plus, leur résolution avec un solveur SAT standard (dans notre cas Minisat) est efficace.

Nous illustrons notre approche avec le problème du Social Golfer : q golfeurs jouent chaque semaine durant w semaines, répartis dans g groupes de p golfeurs

$(q = p.g)$; comment programmer le calendrier de ces golfeurs afin qu'aucun d'eux ne joue plus d'une fois dans le même groupe qu'un autre golfeur ?

Des travaux d'encodage tels que [3] et [5] étudient la relation entre la résolution SAT et CSP (en terme de propriétés telles que la consistance) des contraintes sur les domaines finis. Nous sommes nous intéressés par un autre type de contraintes. Pour le problème du Social Golfer (SGP), le travail le plus proche est [16] qui est une révision et une amélioration de [13]. Alors que ces travaux présente l'encodage direct en SAT, nous nous intéressons à un langage de modélisation de plus haut niveau qui est automatiquement transformé en instances SAT. [16] propose également des techniques pour casser les symétries afin d'améliorer le modèle ; quelques une de ces symétries disparaissent naturellement avec les ensembles (e.g., permutations dues au numérotage des joueurs dans un groupe). Les symétries restantes peuvent être facilement cassées dans notre modèle, soit en ajoutant des contraintes ou en raffinant le modèle.

Il est à noter que nous utilisons la contrainte globale de cardinalité de [4] pour encoder la cardinalité des ensembles.

2 Encodage des contraintes ensemblistes

Nous présentons l'encodage de contraintes ensemblistes classiques en CSP (e.g., $\in$, $\cup$, $\cap$, ...) dans des clauses SAT.

Univers et Supports. De façon informelle, l'univers est l'ensemble des éléments qui sont utilisés dans le modèle d'un certain problème, et le support $\mathcal{F}$ d'un ensemble F de ce modèle est l'ensemble des éléments potentiellement dans F (i.e., $\mathcal{F}$ est un sur-ensemble de F).

Definition 1 *Soit P un problème, et M un modèle de P dans $\mathcal{L}$, i.e., une transcription de P du langage naturel au langage de contraintes $\mathcal{L}$.*

- *l'univers $\mathcal{U}$ de M est un ensemble fini de constantes ;*
- *le support de l'ensemble F du modèle M est un sous-ensemble de l'univers $\mathcal{U}$; nous le notons $\mathcal{F}$. $\mathcal{F}$ représente les éléments de $\mathcal{U}$ qui sont possiblement éléments de F :*

$$F \subseteq \mathcal{F} \subseteq \mathcal{U} \quad et \quad F \in \mathcal{P}(\mathcal{F})$$

où $\mathcal{P}(\mathcal{F}) = \{A | A \subseteq \mathcal{F}\}$ est l'ensemble des parties de $\mathcal{F}$.

Tout élément de $\mathcal{U} \setminus \mathcal{F}$ ne peut faire partie de F. Les majuscules (e.g., F) désignent des ensembles, et les majuscules calligraphiques (e.g., $\mathcal{F}$) leur support. Lorsqu'il n'y a pas de confusion, nous abrégeons "l'ensemble F du modèle M" par "l'ensemble F". Considérons un modèle M avec l'univers $\mathcal{U}$, et un ensemble F sur $\mathcal{F}$. Pour chaque x de $\mathcal{F}$, nous considérons une variable booléenne $x_{\mathcal{F}}$ qui est vraie si $x \in F$ et fausse autrement. Nous appelons l'ensemble de ces variables, les variables support de F. Par la suite, nous écrivons $x_{\mathcal{F}}$ pour $x_{\mathcal{F}} = true$ et $\neg x_{\mathcal{F}}$ pour $x_{\mathcal{F}} = false$.

La règle d'encodage $\Leftrightarrow_{enc}$. Dans la suite, nous considérons 3 ensembles F, G, et H sur leurs supports respectifs $\mathcal{F}$, $\mathcal{G}$ et $\mathcal{H}$ et l'univers $\mathcal{U}$, et pour chaque $x \in \mathcal{U}$ les diverses variables booléennes $x_{\mathcal{F}}$, $x_{\mathcal{G}}$, et $x_{\mathcal{H}}$ comme défini au-dessus. $|G|$ dénote le cardinal de l'ensemble G.

Nous ne forçons pas les supports à être "minimaux" : par exemple, pour la contrainte $F = G$, les ensembles $\mathcal{F} \setminus \mathcal{G}$ et $\mathcal{G} \setminus \mathcal{F}$ peuvent être non vides, alors que $F \setminus G$ et $G \setminus F$ doivent être vides. Ces cas sont considérés par l'encodage. Permettre aux supports de ne pas être minimaux facilite les processus de modélisation. Par contre, l'utilisation de supports réduits limite la taille des instances SAT générées.

Les clauses qui sont générées par la règle d'encodage $\Leftrightarrow_{enc}$ sont de la forme $\forall x \in \mathcal{F},\ \phi(x_{\mathcal{F}})$ qui réprésente les $|\mathcal{F}|$ formules $\phi(x_{\mathcal{F}})$ construites pour chaque élément x du support $\mathcal{F}$ de F (x réfère à l'élément du support/univers, et $x_{\mathcal{F}}$ à la variable representant x pour l'ensemble F). Pour la contrainte d'appartenance, la règle n'est pas quantifiée ; pour les multi-intersection et multi-union, un quantificateur universel additionnel sur i est utilisé pour dénoter un ensemble de règles d'encodage, chacune reliée à un des ensembles $\mathcal{F}_i$.

Le tableau 1 présente nos règles pour les contraintes ensemblistes : d'abord la contrainte, puis son encodage en SAT, et finalement le nombre de clauses générées. Des contraintes telles que $x \notin F$ ou $F \neq G$ sont définies de façon similaire à des contraintes définies dans le tableau 1. De plus, les contraintes peuvent être reliées par les connecteurs $\vee$, $\wedge$, et $\rightarrow$. Dans notre implémentation pour générer les instances SAT, le résultat d'une union doit être stocké dans un ensemble. Ainsi, $H = \bigcup_{i=1}^{n} F_i$ est équivalent à $H = F_1 \cup H_1$, $H_1 = F_2 \cup H_2$, ... La contrainte de multi-union réduit significativement le nombre de variables (pour les ensembles intermédiaires H_i).

3 Encodage SAT pour les modèles à contraintes ensemblistes

Parmi les divers modèles SAT pour le SGP, le plus classique est l'encodage direct (DE) [16] (qui est déjà une révision de [13]). [16] propose également une variante de DE utilisant des variables intermédiaires.

TABLE 1 – Encodage des contraintes ensemblistes en SAT

$x \in F$	$\Leftrightarrow_{enc}$	$x \in \mathcal{F},\ x_{\mathcal{F}}$	1 clause unitaire
		$x \notin \mathcal{F},\ false$	1 clause vide
$F = G$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{F} \cap \mathcal{G},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{G}}$	$2.\lvert\mathcal{F} \cap \mathcal{G}\rvert$ clauses binaires
		$\forall x \in \mathcal{F} \setminus \mathcal{G},\ \neg x_{\mathcal{F}}$	$\lvert\mathcal{F} \setminus \mathcal{G}\rvert$ clauses unitaires
		$\forall x \in \mathcal{G} \setminus \mathcal{F},\ \neg x_{\mathcal{G}}$	$\lvert\mathcal{G} \setminus \mathcal{F}\rvert$ clauses unitaires
$F \cap G = H$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \wedge x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}}$	$\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ clauses ternaires $+ 2.\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ clauses binaires
		$\forall x \in (\mathcal{F} \cap \mathcal{G}) \setminus \mathcal{H},\ \neg x_{\mathcal{F}} \vee \neg x_{\mathcal{G}}$	$\lvert(\mathcal{F} \cap \mathcal{G}) \setminus \mathcal{H}\rvert$ clauses binaires
		$\forall x \in \mathcal{H} \setminus (\mathcal{F} \cap \mathcal{G}),\ \neg x_{\mathcal{H}}$	$\lvert\mathcal{H} \setminus (\mathcal{F} \cap \mathcal{G})\rvert$ clauses unitaires
$F \cup G = H$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \vee x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}}$	$\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ clauses ternaires $+ 2.\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ clauses binaires
		$\forall x \in (\mathcal{F} \cap \mathcal{H}) \setminus \mathcal{G},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{H}}$	$2.\lvert(\mathcal{F} \cap \mathcal{H}) \setminus \mathcal{G}\rvert$ clauses binaires
		$\forall x \in (\mathcal{G} \cap \mathcal{H}) \setminus \mathcal{F},\ x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}}$	$2.\lvert(\mathcal{G} \cap \mathcal{H}) \setminus \mathcal{F}\rvert$ clauses binaires
		$\forall x \in \mathcal{H} \setminus (\mathcal{F} \cup \mathcal{G}),\ \neg x_{\mathcal{H}}$	$\lvert\mathcal{H} \setminus (\mathcal{F} \cup \mathcal{G})\rvert$ clauses unitaires
		$\forall x \in \mathcal{F} \setminus \mathcal{H},\ \neg x_{\mathcal{F}}$	$\lvert\mathcal{F} \setminus \mathcal{H}\rvert$ clauses unitaires
		$\forall x \in \mathcal{G} \setminus \mathcal{H},\ \neg x_{\mathcal{G}}$	$\lvert\mathcal{G} \setminus \mathcal{H}\rvert$ clauses unitaires
$F \subseteq G$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{F} \cap \mathcal{G},\ x_{\mathcal{F}} \rightarrow x_{\mathcal{G}}$	$\lvert\mathcal{F} \cap \mathcal{G}\rvert$ clauses binaires
		$\forall x \in \mathcal{F} \setminus \mathcal{G},\ \neg x_{\mathcal{F}}$	$\lvert\mathcal{F} \setminus \mathcal{G}\rvert$ clauses unitaires
$H = F \setminus G$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \wedge \neg x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}}$	$\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ clauses ternaires $+ 2.\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ clauses binaires
		$\forall x \in \mathcal{F} \setminus (\mathcal{G} \cup \mathcal{H}),\ \neg x_{\mathcal{F}}$	$\lvert\mathcal{F} \setminus (\mathcal{G} \cup \mathcal{H})\rvert$ clauses ternaires
		$\forall x \in \mathcal{H} \setminus \mathcal{F},\ \neg x_{\mathcal{H}}$	$\lvert\mathcal{H} \setminus \mathcal{F}\rvert$ clauses unitaires
		$\forall x \in (\mathcal{F} \cap \mathcal{H}) \setminus \mathcal{G},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{H}}$	$2.\lvert(\mathcal{F} \cap \mathcal{H}) \setminus \mathcal{G}\rvert$ clauses binaires
		$\forall x \in (\mathcal{F} \cap \mathcal{G}) \setminus \mathcal{H},\ x_{\mathcal{F}} \rightarrow x_{\mathcal{G}}$	$\lvert(\mathcal{F} \cap \mathcal{G}) \setminus \mathcal{H}\rvert$ clauses binaires
$H = \bigcup_{i=1}^{n} F_i$	$\Leftrightarrow_{enc}$	$\forall I, J \in \mathcal{P}(N), I \neq \emptyset, I \cup J = N,$ $\forall x \in \mathcal{H} \cap (\bigcap_{i \in I} \mathcal{F}_i) \setminus (\bigcup_{j \in J} \mathcal{F}_j),$ $\bigvee_{i \in I} x_{\mathcal{F}_i} \leftrightarrow x_{\mathcal{H}}$	$\sum_{\substack{I,J \in \mathcal{P}(N),\\ I \neq \emptyset,\\ I \cup J = N}} \left(\lvert\mathcal{H} \cap (\bigcap_{i \in I} \mathcal{F}_i) \setminus (\bigcup_{j \in J} \mathcal{F}_j)\rvert.(\lvert I\rvert + 1)\right)$ clauses binaires et $\sum_{\substack{I,J \in \mathcal{P}(N),\\ I \neq \emptyset,\\ I \cup J = N}} \left(\lvert\mathcal{H} \cap (\bigcap_{i \in I} \mathcal{F}_i) \setminus (\bigcup_{j \in J} \mathcal{F}_j)\rvert\right)$ clauses de tailles $(\lvert I\rvert + 1)$
		$\forall x \in \mathcal{H} \setminus (\bigcup_{i=1}^{n} \mathcal{F}_i),\ \neg x_{\mathcal{H}}$	$\lvert\mathcal{H} \setminus (\bigcup_{i=1}^{n} \mathcal{F}_i)\rvert$ clauses unitaires
		$\forall i \in [1..n],\ \forall x \in \mathcal{F}_i \setminus \mathcal{H},\ \neg x_{\mathcal{F}_i}$	$\sum_{i=1}^{n} \lvert(\mathcal{F}_i \setminus \mathcal{H}\rvert$ clauses unitaires
$H = \bigcap_{i=1}^{n} F_i$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{H} \cap (\bigcap_{i=1}^{n} \mathcal{F}_i),\ \bigwedge_{i=1}^{n} x_{\mathcal{F}_i} \leftrightarrow x_{\mathcal{H}}$	$2.\lvert\mathcal{H} \cap (\bigcap_{i=1}^{n} \mathcal{F}_i)\rvert$ clauses de taille $(\lvert I\rvert + 1)$
		$\forall x \in \bigcap_{i=1}^{n} \mathcal{F}_i \setminus \mathcal{H},\ \bigvee_{i=1}^{n} (\neg x_{\mathcal{F}_i})$	$\lvert\bigcap_{i=1}^{n} \mathcal{F}_i \setminus \mathcal{H}\rvert$ clauses n-aire
		$\forall x \in \mathcal{H} \setminus (\bigcap_{i=1}^{n} \mathcal{F}_i),\ \neg x_{\mathcal{H}}$	$\lvert\mathcal{H} \setminus (\bigcap_{i=1}^{n} \mathcal{F}_i)\rvert$ clauses unitaires
$\lvert G\rvert = k$	$\Leftrightarrow_{enc}$	[4]	$n + \sum_{i=1}^{n} 2u_i^n(\lfloor \frac{u_i^n}{2} \rfloor + 1)(\lceil \frac{u_i^n}{2} \rceil + 1) - (\frac{u_i^n}{2} + 1)$ *clauses et* $\sum_{i=1}^{n} u_i^n$ *variables with* $u_n^n = 1, u_1^n = n$ *and* $u_i^n = u_{2i-1}^n + 2u_{2i}^n + u_{2i+1}^n$.

Nous l ?appellerons TME. Nous proposons un modèle pour le SGP à base de contraintes ensemblistes, indépendantes des solveurs. Ces contraintes sont ensuite encodées en SAT grâce à nos règles $\Leftrightarrow_{enc}$.

Modèle ensembliste Une instance du problème est donnée par un triplet $g-p-w$: p joueurs par groupe, g groupes par semaine, et w semaines. L'univers est l'ensemble des golfeurs $\mathcal{P} = \{p_1, \ldots, p_q\}$ avec $q = g.p$ le nombre total de golfeurs. Nous avons besoin de $w.g$ variables ensemblistes pour modéliser les groupes $G_{1,1}$, ..., $G_{w,g}$. L'ensemble $G_{i,j}$ représente le groupe j de la semaine i et est sur le support $\mathcal{G}_{i,j} = \mathcal{P}$. Chaque $G_{i,j}$ contiendra p joueurs de $\mathcal{P}$. Les supports sont "minimaux" : ils ne peuvent pas être réduits sans perdre de solutions (symétriques). Voici les contraintes du problème du Social Golfer.

— p joueurs par groupe chaque semaine

$$\forall i \in [1..w], \forall j \in [1..g], \quad |G_{i,j}| = p \tag{1}$$

— Chaque golfeur joue chaque semaine

$$\forall i \in [1..w] \quad \bigcup_{j=1..g} G_{i,j} = \mathcal{P} \tag{2}$$

— Un golfeur ne joue pas dans deux groupes la même semaine

$$\forall i \in [1..w] \quad \bigcap_{j=1..g} G_{i,j} = \emptyset \tag{3}$$

Les contraintes (3) ne sont pas nécessaires car elles sont impliquées par les contraintes (1) et (2).

— Deux joueurs ne peuvent pas jouer deux fois dans le même groupe

$$\begin{aligned} &\forall w_1, w_2 \in [1..w], p_i, p_j \in \mathcal{P}, \\ &g_1, g_2 \in [1..g], w_1 > w_2 \ \wedge \ i > j \ \wedge \\ &p_i \in G_{w_1,g_1} \ \wedge p_j \in G_{w_1,g_1} \\ &\wedge \ p_i \in G_{w_2,g_2} \ \rightarrow \ p_j \notin G_{w_2,g_2} \end{aligned} \tag{4}$$

Une autre formulation peut être donnée en utilisant les contraintes de cardinalité :

$$\begin{aligned} &\forall w_1, w_2 \in [1..w], g_1, g_2 \in [1..g], \\ &w_1 > w_2 \ \wedge \ g_1 \geq g_2 \ \wedge \\ &|G_{w_1,g_1} \cap G_{w_2,g_2}| \leq 1 \end{aligned} \tag{5}$$

SCE : encodage des contraintes ensemblistes A partir du modèle ci-dessus, nos règles $\Leftrightarrow_{enc}$ d'encodage vont automatiquement générer l'instance SAT correspondante (voir section 2). Le nombre de clauses générées est :

— Les contraintes (1) génèrent $w.g.w.(g.p + \sum_{i=1}^{g.p}[2u_i^{g.p}(\lfloor \frac{u_i^{g.p}}{2} \rfloor + 1)(\lceil \frac{u_i^{g.p}}{2} \rceil + 1) - (\frac{u_i^{g.p}}{2} + 1)])$ clauses avec $u_{g.p}^{g.p} = 1, u_1^{g.p} = g.p$ et $u_i^{g.p} = u_{2i-1}^{g.p} + 2u_{2i}^{g.p} + u_{2i+1}^{g.p}$. La complexité de la formule générée par les contraintes (1) est $\mathcal{O}(w^2.g^3.p^2)$.

— Les contraintes (2) génèrent $w.g.p$ clauses.

— Les contraintes (4) génèrent $w.(w-1).g.(g+1).q.(q-1)/2)$ clauses $(\mathcal{O}(w^2.g^4.p^2))$ et les contraintes (5) génèrent $w.((w-1)/2).g.((g+1)/2).3.q.(q + \sum_{i=1}^{q}[2u_i^q(\lfloor \frac{u_i^q}{2} \rfloor + 1)(\lceil \frac{u_i^q}{2} \rceil + 1) - (\frac{u_i^q}{2} + 1)])$ clauses $(\mathcal{O}(w^2.g^5.p^3))$.

Complexité des instances SAT Puisque la complexité des contraintes (4) est $\mathcal{O}(w^2.g^4.p^2)$ alors que la complexité des contraintes (5) est $\mathcal{O}(w^2.g^5.p^3)$, nous nous concentrerons sur la formulation avec implications (contraintes (4)). La complexité des instances SAT générées par l'encodage du modèle par contraintes ensemblistes (SCE) constitué des contraintes (1), (2), et (4) est $\mathcal{O}(w^2.g^4.p^2)$.

Post-traitement par propagation unitaire La propagation unitaire est un procédé simple pour éliminer des clauses unitaires (des clauses avec uniquement des littéraux faux et un littéral libre) en donnant la valeur vrai aux littéraux libres. Cette valuation peut donner de nouvelles clauses unitaires, et le procédé est alors itéré jusqu'à atteindre un point fixe. Les algorithmes de propagation unitaire peuvent significativement réduire : 1) la taille des instances, 2) le nombre de variables et 3) le temps de résolution. Il est à noter que l'encodage que nous avons choisi pour la contrainte de cardinalité génère de nombreuses clauses unitaires qui disparaissent avec la propagation unitaire.

4 Suppression de symétries pour le SGP

Retirer des solutions symétriques facilite le travail d'un solveur complet. Le problème du Social Golfer possède de nombreuses symétries : la position d'un joueur dans un groupe n'est pas pertinente ; les groupes d'une semaine peuvent être renumérotés ; les semaines peuvent être permutées, ... La suppression de symétries consiste à éliminer les symétries par l'ajout de nouvelles contraintes ou la modification du modèle. [13] propose des clauses pour retirer les symétries entre les joueurs, pour ordonner les groupes d'une semaine, et pour ordonner les semaines. Cependant, ces clauses deviennent de plus en plus complexes et des erreurs peuvent facilement s'introduire. En fait, [16] corrige les clauses pour la suppression de symétries de [13] (erreurs d'indices principalement dans les nombreux opérateurs $\bigvee$ et $\bigwedge$). Nous appelons cet encodage TME^{SB}. D'autres symétries peuvent être supprimées

([11] ou [10]). Toutes les symétries peuvent être supprimées [7], mais au coût d'un nombre super exponentielle de contraintes qui ne peut pas être considéré en pratique.

Avec notre langage nous pouvons supprimer les symétries en ajoutant de nouvelles contraintes au modèle initial ou en raffinant ce modèle par modification de supports et contraintes ; Notre modèle étant différent de celui de [16], nous n'avons pas les mêmes symétries. Nous essayons cependant de supprimer les similaires. Le premier groupe de suppression de symétries ($SB1$) consiste à remplir la première semaine : les p premiers joueurs dans le 1er groupe, les p suivants dans le second, et ainsi de suite. Afin de complèter $SB1$, nous considérons le groupe $SB2$: les p premiers joueurs (qui ont déjà joué ensemble dans le 1er groupe de la 1ère semaine grâce à $SB1$) sont répartis dans différents groupes chaque semaine. Quand p est plus grand que g il est clair que le problème n'a pas de solution. Par la suite nous considérons donc que $g \geq p$.

Suppression de symétries pour le modèle ensembliste par ajout de contraintes Pour $SB1$, il suffit d'ajouter les contraintes suivantes au modèle SCE.

$$\forall i \in [1..p.g], p_i \in G_{1,i \ div \ (p+1)} \tag{6}$$

Pour le 2ème groupe $SB2$ de suppression de symétries, les contraintes nécessaires sont également simples :

$$\forall i \in [2..w], \forall j \in [1..p], p_j \in G_{i,j} \tag{7}$$

Ces contraintes augmentent le nombre de clauses des instances SAT générées, mais toutes ces nouvelles clauses sont unitaires et vont donc être utilisées (puis disparaître) par propagation unitaire. L'encodage SAT de ce modèle ensembliste avec suppression de symétries par ajout de contraintes est nommé SCE^{SBC} et consiste en les contraintes (1), (2), (4), (6) et (7).

Suppression de symétries par modification du modèle. Bien que cela soit plus ennuyeux, cela réduit les supports des ensembles et les cardinalités, et par conséquent la taille des instances SAT générées. Pour $SB1$ l'unique modification consiste à modifier le support des groupes de la 1ère semaine et à fixer ses groupes :

$$\begin{aligned} \forall i \in [1..g], \mathcal{G}_{1,i} = \{p_{1+(i-1).g}, \dots p_{p+(i-1).g}\} \\ \text{and} \ \ \forall i \in [1..g], G_{1,i} = \mathcal{G}_{1,i} \end{aligned} \tag{8}$$

Les autres ensembles, variables, et contraintes restent inchangés. Pour $SB2$, nous changeons les variables de groupes. Plutôt que les $G_{i,j}$, nous considérons les ensembles $G'_{1,1}, \dots, G'_{w,g}$ tels que :

— pour la 1ère semaine $G_{i,j} = G'_{i,j}$;
— pour les semaines suivantes $G_{i,j} = G'_{i,j} \cup \{p_j\}$ si $j \leq p$, $G_{i,j} = G'_{i,j}$ sinon.

Le support des $G'_{1,i}$ (i.e., les groupes de la 1ère semaine) sont définis comme avec $SB1$. Puisque les p premiers joueurs sont répartis sur les p premiers groupes de chaque semaine, les supports des autres groupes peuvent être réduits à $\mathcal{P}' = \{p_{p+1}, \dots, p_q\}$, et $\forall i \in [2..w], \forall j \in [1..g], \mathcal{G}_{i,j} = \mathcal{P}'$

P joueurs par groupe chaque semaine. Les contraintes (1) doivent être remplacées par les contraintes (9)–(11).

$$\forall i \in [1..g], \ \ |G'_{1,i}| = p \tag{9}$$

$$\forall j \in [2..w], \forall i \in [1..p], \ \ |G'_{j,i}| = p - 1 \tag{10}$$

$$\forall j \in [2..w], \forall i \in [p+1..g], \ \ |G'_{j,i}| = p \tag{11}$$

Chaque joueur joue chaque semaine. Les contraintes (12) remplacent les contraintes (2).

$$\forall j \in [2..w] \ \bigcup_{i=1..g} G_{j,i} = \mathcal{P}' \tag{12}$$

Deux joueurs ne peuvent pas jouer deux fois dans le même groupe. Les contraintes (4) sont remplacées par les contraintes (13)–(17). Puisque 2 groupes $G_{i,j}$ avec $j \leq p$ et $i > 1$ ont le joueur p_j en commun, les groupes correspondants $G'_{i,j}$ (dont les supports ne contiennent pas les p_l, $l \leq p$) ne peuvent pas avoir d'autres joueurs p_k en commun :

$$\begin{aligned} \forall w_1, w_2 \in [2..w], \ p_i \in \mathcal{P}, \ g_1 \in [1..p], \ w_1 > w_2, \\ p_i \in G'_{w_1,g_1} \ \rightarrow \ p_i \notin G'_{w_2,g_1} \end{aligned} \tag{13}$$

La relation entre les autres paires de groupes ne change pas.
Entre un groupe de la 1ère semaine (sauf le 1er groupe) et les groupes des autres semaines :

$$\begin{aligned} \forall w_1 \in [2..w], \ p_i, p_j \in \mathcal{P}, \ g_1 \in [2..g], g_2 \in [1..g], \\ i > j, p_i \in G'_{1,g_1} \ \wedge \ p_j \in G'_{1,g_1} \ \wedge \\ p_i \in G'_{w_1,g_2} \ \rightarrow \ p_j \notin G'_{w_1,g_2} \end{aligned} \tag{14}$$

Entre deux groupes (sauf de la 1ère semaine) de même numéro avec un indice supérieur à p :

$$\begin{aligned} \forall w_1, w_2 \in [2..w], \ p_i, p_j \in \mathcal{P}, \ g_1 \in [p+1..g], \\ w_1 > w_2, \ i > j, p_i \in G'_{w_1,g_1} \wedge p_j \in G'_{w_1,g_1} \wedge \\ p_i \in G'_{w_2,g_1} \ \rightarrow \ p_j \notin G'_{w_2,g_1} \end{aligned} \tag{15}$$

Entre deux groupes (sauf de la 1ère semaine) de numéros différents :

$$\begin{aligned} \forall w_1, w_2 \in [2..w], \ p_i, p_j \in \mathcal{P}, \ g_1, g_2 \in [1..g], \\ w_1 > w_2, g_1 \neq g_2, \ i > j, p_i \in G'_{w_1,g_1} \ \wedge \\ p_j \in G'_{w_1,g_1} \ \wedge \ p_i \in G'_{w_2,g_2} \ \rightarrow \ p_j \notin G'_{w_2,g_2} \end{aligned} \tag{16}$$

L'encodage SAT du modèle ensembliste avec suppression de symétries par modification de modèle est nommé SCE^{SBM} et est composé des contraintes (8)–(17).

5 Comparaisons de modèles

Le tableau 2 résume les codages (décrits auparavant) que nous comparons dans la prochaine section. NOM_{UP} dénote le codage NOM après propagation unitaire.

Expressivité. Les variables de notre modèle ensembliste sont bien plus simples : 2 indices au lieu de 4, ceci les rendant plus lisibles. En fait, nous n'avons pas à numéroter les positions dans un groupe, et nous n'avons pas à ajouter un index pour représenter le numéro du joueur. La seconde différence est la simplicité et la lisibilité des contraintes. De fait, les contraintes ensemblistes sont plus expressives que de pures clauses SAT. Ensuite, l'encodage en SAT est fait par les règles $\Leftrightarrow_{enc}$. L'avantage est double : 1) les contraintes sont lisibles, expressives, faciles à modifier, et le modèle qui en résulte est donc plus compréhensible ; 2) moins de fautes peuvent s'introduire car le procédé de modélisation est beaucoup plus simple et léger. De plus, le modèle ensembliste est indépendant des solveurs : le même modèle (exceptée l'exacte syntaxe) peut être résolu pas un solveur CSP sur les ensembles ou par un solveur SAT après encodage par les règles $\Leftrightarrow_{enc}$.

En résumé, en terme d'expressivité, lisibilité, sensibilité aux erreurs, et indépendance aux solveurs, notre modèle ensembliste est supérieur à un encodage direct tel que DE ou TME. La suppression de symétries est également plus simple dans notre modèle.

Structure du modèle Afin de comparer les encodages, nous avons généré des instances du SGP avec : l'encodage direct DE, celui de Triska-Musliu [16] (TME), et notre encodage de modèle ensembliste avec propagation unitaire (SCE_{UP}) et sans (SCE). Dans le tableau 3, chaque instance est définie par le triplet (groupes, joueurs par groupe, semaines) et pour chaque encodage, le nombre de clauses et variables (générées). L'encodage le plus petit pour chaque instance est en gras. On ne peut pas comparer les encodages uniquement par leur taille. Cependant, les très grosses instances ne sont pas résolvables, due aux limites de mémoires. Il est donc primordial de générer des instances les plus petites possibles.

L'encodage direct (DE) est clairement inutilisable quand le nombre de joueurs ou de groupes grossit : le nombre de clauses explose immédiatement. Avec l'introduction de variables intermédiaires, le nombre de clauses est moins important pour TME, mais le nombre de variables grossit tout de même. SCE produit plus de variables mais moins de clauses. Comme supposé, SCE_{UP} produit l'encodage le plus intéressant en terme de variables et clauses : en fait, SCE génère de nombreuses clauses unitaires ou binaires (section 3) qui disparaissent après propagation unitaire.

Impact de la suppression de symétries Nous avons appliqué les 2 groupes de suppression de symétries présentés en section 4 ; les résultats sont présentés dans le tableau 3. Pour TME, l'introduction de contraintes supplémentaires augmente le nombre de clauses (environ 10% de clauses supplémentaires), et le nombre de variables ne change pas. Il est à noter que la propagation unitaire est inutile pour les instances TME et TME^{SB}, car ces instances ne possèdent pas de clauses unitaires. Pour SCE, la suppression de symétries par ajout de contraintes accroit de façon négligeable le nombre de contraintes (voir SCE^{SBC}). La suppression de symétrie par modification du modèle (SCE^{SBM}) réduit significativement la taille des instances SAT générées : de 20 à 60% de variables en moins et de 40 à 60% de clauses en moins. Cette réduction significative est due à la réduction des supports et aux contraintes de cardinalité.

Sans propagation unitaire, les instances de SCE^{SBM} sont toujours les plus petites en terme de nombre de clauses. La propagation unitaire n'a aucun impact sur TME. Cependant son impact est significatif sur SCE, SCE^{SBM} et SCE^{SBC}. Pour SCE, la propagation unitaire divise le nombre de variables par un facteur de 6 à 25 : ceci est principalement dû aux variables des contraintes de cardinalité. Le nombre de clauses est réduit d'environ 10%. Pour SCE^{SBC}, la propagation unitaire réduit encore plus le nombre de variables (jusqu'à 30 fois moins de variables). Le nombre de clauses est lui réduit de 30 à 60%. Pour SCE^{SBM}, la propagation unitaire est moins spectaculaire : en fait, le modèle intiale est lui-même réduit. Cependant, le nombre de variables est divisé par un facteur de 5 à 15. Le nombre de clauses est réduit d'environ 10%.

En résumé, la propagation unitaire est plus bénéfique à SCE^{SBC} ; cependant, $\text{SCE}^{\text{SBM}}_{\text{UP}}$ donne toujours les meilleures instances en terme de nombre de variables et clauses.

6 Analyse expérimentale

Nous comparons maintenant l'efficacité des encodages en terme de temps de résolution, et pour cela, nous utilisons le solveur MiniSat [9]. SatELite [8] est un pré-traitement de Minisat qui réduit énormément

TABLE 2 – Liste des encodages

Nom	Description	Contraintes
DE	Encodage direct	[16]
TME	encodage Triska-Musliu	[16]
TME^{SB}	TME avec suppression de symétries	[16]
SCE	encodage SAT du modèle ensembliste	(1), (2), (4)
SCE^{SBC}	SCE avec suppression de symétries par ajout de contraintes	(1), (2), (4),(6), (7)
SCE^{SBM}	SCE avec suppression de symétries par modification de modèle	(8)–(17)
NAME_{UP}	encodage après propagation unitaire	

TABLE 3 – Taille des instances générées avec différents encodages.

	DE		TME		TME_*^{SB}		SCE		SCE^{SBM}		SCE^{SBC}		SCE_{UP}		$\text{SCE}_{\text{UP}}^{\text{SBM}}$		$\text{SCE}_{\text{UP}}^{\text{SBC}}$	
Prob.	var	cl. $\times 10^6$	var	cl. $\times 10^3$	var	cl. $\times 10^3$	var	cl. $\times 10^3$	var	cl. $\times 10^3$	var	cl. $\times 10^3$	var	cl. $\times 10^3$	var	cl. $\times 10^3$	var	cl. $\times 10^3$
5-3-6	1 350	3	1 800	60	1 800	71	8 625	50	5 702	21	8 625	50	1 410	44	**860**	**18**	980	23
5-3-7	1 575	4	2 100	79	2 100	92	11 110	68	7 734	30	11 110	68	1 645	60	**1 032**	**26**	1 176	34
8-4-4	4 096	48	5 120	323	5 120	390	24 224	235	14 192	96	24 224	235	3 840	205	**2 376**	**78**	2 580	92
8-4-5	5 120	81	6 400	483	6 400	567	34 752	373	22 476	173	34 752	373	4 800	335	**3 168**	**149**	3 440	176
8-4-6	6 144	121	7 680	675	7 680	776	47 072	543	32 552	273	47 072	543	5 760	498	**3 960**	**243**	4 300	288
8-4-7	7 168	170	8 960	898	8 960	1 016	61 184	744	44 420	396	61 184	744	6 720	692	**4 752**	**361**	5 160	427
8-4-8	8 192	227	10 240	1 153	10 240	1 288	77 088	978	58 080	542	77 088	978	7 680	918	**5 544**	**500**	6 020	593
8-4-9	9 216	292	11 520	1 441	11 520	1 592	94 784	1 243	73 532	711	94 784	1 243	8 640	1 175	**6 336**	**663**	6 880	786
8-4-10	10 240	365	12 800	1 759	12 800	1 928	114 272	1 540	90 776	902	114 272	1 540	9 600	1 465	**7 128**	**848**	7 740	1 006
9-4-6	7 776	196	9 720	1 048	9 720	1 191	117 324	858	46 344	448	117 324	858	7 344	793	**5 620**	**472**	5 620	472
9-4-7	9 072	274	11 340	1 401	11 340	1 568	157 284	1 180	63 368	652	157 284	1 180	8 568	1 104	**6 008**	**562**	6 744	701
9-4-8	10 368	366	12 960	1 805	12 960	1 996	203 076	1 553	82 984	895	203 076	1 553	9 792	1 465	**7 024**	**783**	7 868	975
9-4-9	11 664	470	14 580	2 261	14 580	2 261	254 700	1 976	105 192	1 176	254 700	1 977	11 016	1 878	**8 040**	**1 040**	8 992	1 294
9-4-10	12 960	588	16 200	2 767	16 200	3 006	312 156	2 451	129 992	1 496	312 156	2 451	12 240	2 342	**9 056**	**1 334**	10 116	1 658

le nombre de clauses (e.g., en utilisant de la détection de subsomptions) et variables (e.g., en éliminant les littéraux purs). Ce pré-traitement a un coût, mais il améliore généralement le temps global de résolution. Il peut aussi être désactivé. Le tableau 4 représente les temps de résolution de Minisat, avec ou sans le pré-traitement SatElite.

Les expérimentations sont réalisées avec un CPU Intel Core i5-2540M cadencé à 2.60GHz CPU et 4 GB RAM. Pour chaque résolution, un temps limite de 300 secondes est accordé ("-" si le temps limite est expiré). De plus longs temps d'execution on été testés, mais aucune différence significative n'a été observée. Les résultats pour l'encodage DE ne sont pas présentés car, comme supposé, aucun résultats n'a été otenu dans un temps raisonnable.

Le tableau 4 montre que l'utilité de SatElite est difficile à prédire : selon les instances, il peut significativement améliorer ou détériorer les résultats. En moyenne, il n'améliore pas les résultats, et les meilleurs temps de résolution sont obtenus sans ce pré-traitement. La suppression de symétries par modification du modèle (SCE^{SBM}) donne les meilleurs (ou très proche du meilleur) résultats, avec ou sans pré-traitement. La propagation unitaire a peu d'influence sur les temps de résolution pour l'encodage SCE^{SBM}. La suppression de symétries par ajout de contraintes (SCE^{SBC}) n'améliore pas, sauf quand la propagation unitaire est appliquée ($\text{SCE}_{\text{UP}}^{\text{SBC}}$). En fait, $\text{SCE}_{\text{UP}}^{\text{SBC}}$ obtient des résultats aussi bons que SCE^{SBM}. Supprimer les symétries dans TME est plutôt un procédé fluctuant : selon les instances et l'usage ou non de SatELite, les résultats sont significativement améliorés ou dégradés.

En résumé, les meilleurs résultats sont obtenus avec notre modèle contraintes ensemblistes, avec $\text{SCE}_{\text{UP}}^{\text{SBC}}$ quand le pré-traitement est appliqué, ou de manière prédominante avec $\text{SCE}_{\text{UP}}^{\text{SBM}}$ quand le pré-traitement n'est pas appliqué. Finalement, les meilleurs résultats

TABLE 4 – Temps de résolution Minisat

Prob.	TME	TME^{SB}	SCE	SCE^{SBM}	SCE^{SBC}	SCE_{UP}	$\text{SCE}^{\text{SBM}}_{\text{UP}}$	$\text{SCE}^{\text{SBC}}_{\text{UP}}$	TME	TME^{SB}	SCE	SCE^{SBM}	SCE^{SBC}	SCE_{UP}	$\text{SCE}^{\text{SBM}}_{\text{UP}}$	$\text{SCE}^{\text{SBC}}_{\text{UP}}$
	avec SatElite								**sans** SatElite							
5-3-6	8.92	0.69	0.18	0.06	0.12	0.12	0.07	**0.04**	9.37	0.30	1.05	**0.01**	**0.01**	0.26	**0.01**	**0.01**
5-3-7	98.28	13.37	1.42	0.13	1.21	5.09	0.09	**0.08**	97.47	24.86	9.19	**0.06**	0.13	5.67	1.79	0.28
8-4-4	1.04	1.33	0.97	0.32	1.19	0.90	0.29	**0.27**	0.05	0.23	0.09	**0.03**	0.07	0.07	**0.03**	**0.03**
8-4-5	2.26	2.64	1.93	0.86	2.51	1.89	0.84	**0.78**	0.08	0.58	0.13	0.06	0.11	0.06	**0.05**	0.07
8-4-6	4.44	5.16	3.65	1.87	4.74	3.65	1.82	**1.71**	0.25	3.58	0.27	0.14	0.18	0.19	**0.08**	0.09
8-4-7	34.25	94.68	8.66	3.59	8.52	7.52	3.64	**3.46**	27.05	25.88	3.53	**0.48**	1.71	1.94	0.56	0.98
8-4-8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8-4-9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8-4-10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9-4-6	8.45	10.52	11.24	3.15	10.34	11.10	**2.71**	4.58	0.23	3.72	0.37	0.13	0.29	0.25	**0.11**	0.13
9-4-7	13.69	27.16	18.95	5.80	17.8	19.04	**5.12**	8.76	0.31	6.61	0.58	0.22	0.51	0.36	**0.14**	0.24
9-4-8	-	-	31.87	**11.10**	29.60	31.48	12.72	14.90	247.83	-	14.66	5.03	1.10	20.93	2.62	**0.68**
9-4-9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9-4-10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

sont obtenus sans pré-traitement.

7 Discussion et Conclusion

Modélisation. Modéliser un problème avec des contraintes ensemblistes et ensuite générer automatiquement l'instance SAT correspondante est beaucoup plus simple que d'écrire directement les instances SAT comme avec DE ou TME. La suppression des symétries est plutôt ardue dans les encodages directs, et très simple par ajout de contraintes dans le modèle ensembliste (un peu plus ardue par rafinement de modèle). Utiliser un formalisme de haut niveau est donc bénéfique à la phase de modélisation : cela simplifie la tâche, et limite les possibilités d'introduction d'erreurs (généralement dues à la manipulation de nombreux indices). L'encodage SAT est ensuite automatisé.

Instances SAT Nous avons montré que les instances SAT générées automatiquement par notre encodage sont de bonne qualité : 1) elles contiennent toujours significativement moins de clauses (avec ou sans suppression de symétrie, et avec ou sans propagation unitaire) ; 2) après propagation unitaire, elles contiennent également moins de variables ; et 3) elles sont résolues plus efficacement par Minisat, sans "tuning de paramètres", avec ou sans pré-traitement par SatElite.

Suppression de symétries Nous avons montré que supprimer des symétries par ajout de contraintes au modèle ensembliste est très simple. De plus, après propagation unitaire, les instances SAT générées sont beaucoup plus petites, et leur temps de résolution est également amélioré. La suppression de symétries par modification du modèle est encore plus bénéfique. L'effort supplémentaire par rapport à l'ajout de contraintes, est très bénéfique au niveau de la taille des instances, mais juste valable en terme de temps de résolution (ceci dépend des instances et du prétraitement). Ainsi, il faut mettre en balance temps de résolution et de modélisation. La taille des instances générées peut aussi être un facteur décisif : de plus gros problèmes peuvent être générés par la suppression de symétries en modifiant le modèle (voir SCE^{SBM}).

Contraintes ensemblistes en programmation par contraintes En terme d'ensembles, l'expressivité de notre proposition est plus ou moins similaire à celle d'un système de programmation par contraintes tel que [1] : c'était notre but. L'avantage de [14] ou [1], est que les contraintes ensemblistes peuvent être mélangées à d'autres contraintes. Nous prévoyons aussi l'encodage d'autres contraintes dans le futur. Dans des sytèmes tels que [14], un solveur spécial doit être réalisé (à base de réduction de domaine et d'énumération). [2] compare des solveurs CSP ensemblistes pour le SGP : la plupart des résultats sont obtenus par des heuristiques de recherche (dynamiques) ou des mécanismes spécifiques de résolution. Notre approche est très différente : nous ne voulons pas réaliser un solveur spécifique, ni adapter ou régler un solveur existant pour résoudre efficacement nos instances SAT ; nous voulons transformer un modèle ensembliste de haut niveau en une instance SAT de "bonne" qualité (en terme de taille et résolution) qui est efficacement résolue par un solveur SAT classique.

En terme de résolution, l'instance 5-3-6 de notre modèle ensembliste n'est pas résolue avant le temps limite par le solveur standard de [1], alors que son encodage est résolu en moins d'une seconde par Minisat.

Afin de réduire encore la taille des instances SAT générées, nous prévoyons d'ajouter un pré-traitement (forcer une consistance locale) pour réduire les supports des ensembles. Pour le SGP, les supports sont "minimaux" (en terme de réduction comme dans [14]). Cependant, pour d'autres problèmes, les supports peuvent être réduits par un processus de déduction (sans perdre de solution), et donc, les instances SAT générées peuvent également être plus petites. Un tel mécanisme pourrait être équivalent à une phase de réduction du système [14].

Conclusion Nous avons présenté une technique pour l'encodage en SAT de contraintes ensemblistes : la modélisation est effectuée en utilisant des contraintes ensemblistes expressives et déclaratives qui sont ensuite automatiquement converties en variables et clauses SAT grâce à nos règles d'encodage $\Leftrightarrow_{enc}$. Cette technique a été appliquée avec succès au problème du Social Golfer, et à la suppression de symétries pour ce problème. Les avantages de notre technique sont les suivants : 1) la modélisation est expressive, déclarative et lisible. De plus, les modèles sont indépendants du solveur et peuvent être résolus par un solveur CSP ou SAT. 2) cette technique limite l'introduction d'erreurs, ce qui est fréquent avec des codages directs en SAT ; 3) des symétries peuvent être supprimées par le simple ajout de contraintes supplémentaires ou par raffinement du modèle ; 4) les instances SAT qui sont automatiquement générées sont plus petites que celles de [16] qui sont directement écrites et ont déjà subi des améliorations ; après propagation unitaire, nos instances contiennent également moins de variables que celles de [16] ; 5) finalement, par rapport au temps de résolution, nos instances du SGP automatiquement générées sont résolues plus rapidement, avec ou sans propagation unitaire, avec ou sans suppression de symétries et avec ou sans SatElite.

Nous avons appliqué notre technique à d'autres problèmes (tels que le car-sequencing, les n-reines, le sudoku, ...). Nous obtenons toujours des modèles très simples et très lisibles. Les instances SAT générées semblent également convenir parfaitement à un solveur tel que Minisat.

Nous avons l'intention d'adapter notre encodage pour d'autres domaines, tels que les domaines finis et les séquences. Nous prévoyons également de raffiner la notion de supports et d'intégrer un pré-traitement afin de les réduire. Cela n'aura pas d'impact sur le modèle du SGP que nous avons proposé, mais pour beaucoup de problèmes, cela permettra d'obtenir des instances SAT encore plus petites.

Références

[1] Minizinc. http ://www.minizinc.org/.

[2] F. Azevedo. An attempt to dynamically break symmetries in the social golfers problem. In *CSCLP*, pages 33–47, 2006.

[3] F. Bacchus. Gac via unit propagation. In *Proc. of CP 2007*, volume 4741 of *LNCS*, pages 133–147. Springer, 2007.

[4] O. Bailleux and Y. Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *Proc. of CP 2003*, volume 2833, pages 108–122. Springer, 2003.

[5] C. Bessière, E. Hebrard, and T. Walsh. Local consistencies in sat. In *Selected Revised Papers of SAT 2003.*, volume 2919 of *LNCS*, pages 299–314. Springer, 2004.

[6] C. Cotta, I. Dotú, A. J. Fernández, and P. V. Hentenryck. Scheduling social golfers with memetic evolutionary programming. In *HM 2006*, volume 4030 of *LNCS*, pages 150–161. Springer, 2006.

[7] J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proc. of KR'96*, pages 148–159. Morgan Kaufmann, 1996.

[8] N. Eén and A. Biere. Effective preprocessing in sat through variable and clause elimination. In *SAT 2005*, volume 3569, pages 61–75, 2005.

[9] N. Eén and N. Sörensson. An extensible sat-solver. In *SAT 2003*, vol. 2919, pages 502–518, 2003.

[10] P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *CP 2002*, vol. 2470, pages 462–476. Springer, 2002.

[11] A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographic orderings. In *Proc. of CP 2002*, volume 2470, pages 93–108. Springer, 2002.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman & Company, San Francisco, 1979.

[13] I. Gent and I. Lynce. A sat encoding for the social golfer problem. In *IJCAI'05 workshop on modelling and solving problems with constraints*, 2005.

[14] C. Gervet. Conjunto : Constraint propagation over set constraints with finite set domain variables. In *Proc. of ICLP'94*. MIT Press.

[15] F. Rossi, P. van Beek, and T. Walsh, eds. *Handbook of Constraint Programming.* Elsevier, 2006.

[16] M. Triska and N. Musliu. An improved sat formulation for the social golfer problem. *Annals of Operations Research*, 194(1) :427–438, 2012.

Comparaison de méthodes de résolution pour le problème de somme coloration

Clément Lecat, Chu Min Li, Corinne Lucet, Yu Li

Université de Picardie Jules Verne, Laboratoire MIS
Pôle Sciences, 33 rue St Leu, 80000 Amiens

{clement.lecat, chu-min.li, corinne.lucet, yu.li}@u-picardie.fr

Résumé

Le problème de somme coloration (*MSCP*) est un problème dérivé du problème de coloration (*GCP*). Pour ces deux problèmes NP-difficiles, il s'agit d'affecter une couleur c_i à chaque sommet v d'un graphe $G = (V, E)$, en respectant les contraintes de voisinage : deux sommets voisins ne peuvent avoir une couleur identique. Pour *MSCP*, nous considérons qu'un poids p_i est associé à chaque couleur c_i. Alors que *GCP* a pour but de *minimiser le nombre des couleurs utilisées*, *MSCP* a pour but de *minimiser la somme des poids des couleurs utilisées*.

À travers ce papier, nous présentons et comparons quatre méthodes exactes permettant de résoudre le problème *MSCP*. La première de ces méthodes est une méthode de séparation et évaluation, basée sur le calcul de bornes supérieures et inférieures. Les deux méthodes suivantes consistent à transformer le graphe en une instance *MaxSAT* et en une instance *MinSAT*, et de les résoudre avec un solveur approprié. La dernière méthode consiste à transformer le problème *MSCP* en un problème COP et à utiliser un solveur dédié à la résolution.

Abstract

The Minimum Sum Coloring Problem (*MSCP*) is an extension to the well known Graph Coloring Problem (*GCP*). Both NP-hard problems consist in assigning a color c_i to each vertex v of a graph $G = (V, E)$ while respecting the neighborhood constraints : two adjacent vertices cannot share a same color. For *MSCP*, a weight p_i is associated with each color c_i. While the objective of *GCP* is *to minimize the number of colors*, the objective of *MSCP* is *to minimize the sum of the color weights*.

In this paper, we propose and compare four methods to solve *MSCP*. The first one is a Branch-and-Bound method which uses upper and lower bounds to reduce search space. The second, the third and the fourth methods encode $MSCP$ instances into a $MaxSAT$, $MinSAT$ and COP instance respectively, and solve these instances with dedicated solvers.

1 Introduction

Le problème de coloration de graphe (Graph Coloring Problem *GCP*) est un problème majeur en optimisation combinatoire. De nombreuses études ont été menées autour de ce problème NP-difficile [7]. Les méthodes proposées pour résoudre ce problème sont multiples et se divisent en deux catégories, les méthodes exactes et les méthodes approchées. Les méthodes exactes visent à fournir une solution optimale au problème, elles regroupent les méthodes de Branch-and-Bound (*e.g.* [22]), de décomposition de graphe (*e.g.* [20]), mais aussi les formulations sous forme de problème SAT [26]. Les méthodes approchées fournissent une borne supérieure ou inférieure au problème et rassemblent les algorithmes gloutons comme le célèbre DSATUR [5] et des heuristiques et méta-heuristiques [24, 8, 19]. Dans la littérature ces méthodes sont généralement testées sur des benchmarks de référence comme DIMACS et COLOR [6, 10].

Le problème de somme coloration minimale d'un graphe (Minimal Sum Coloring Problem *MSCP*) est un problème dérivé de *GCP* qui fut introduit en 1989 par Kubicka et Schwenk [14]. Les principaux résultats publiés sur *MSCP* montrent des propriétés structurelles relatives aux familles de graphes pour lesquelles il existe des algorithmes efficaces et des bornes théoriques [11, 1, 21, 3]. Des études plus récentes proposent des heuristiques [18] et méta-heuristiques [25, 12, 4, 23] qui fournissent des bornes

pour les instances ci-dessus citées.

Les travaux que nous reportons dans cet article, portent principalement sur une approche exacte pour le problème *MSCP*, dédiées donc à produire une solution optimale pour ce problème. Nous proposons une méthode de Branch-and-Bound basée sur l'élaboration de bornes supérieures et inférieures pertinentes, et un encodage du problème *MSCP* en une instance $MaxSAT$, $MinSAT$ et COP. Nous analysons et comparons ensuite ces quatre méthodes exactes pour MSCP. Les résultats ont été obtenus sur des instances structurées des benchmarks de référence et sur des graphes aléatoires.

Ce papier est organisé de la façon suivante : Dans la section 2 nous donnons une définition formelle des problèmes *GCP* et *MSCP*. Dans la section 3, nous présentons le calcul de bornes inférieures (LB). Dans la section 4, nous présentons la transformation d'un problème *MSCP* en un problème *MaxSAT* puis en un problème *MinSAT* et pour terminer cette section, en un problème COP. Enfin dans la section 5, nous comparons le temps de réponse de ces différentes méthodes sur un ensemble d'instances.

2 Problème de somme coloration

2.1 Définitions préliminaires

Soit un graphe non orienté *G=(V,E)* où V est l'ensemble des sommets ($|V| = n$) et $E \subseteq V \times V$ l'ensemble des arêtes ($|E| = m$). Le voisinage $\mathcal{N}(v)$ d'un sommet $v \in V$ est défini comme suit : $\mathcal{N}(v) = \{u \in V \mid (v,u) \in E\}$. Le degré d'un sommet v est le nombre de sommets voisins de v, nous le notons $d(v)$. Le degré du graphe G est $\Delta(G) = max\{d(v); \forall v \in V\}$. Un *sous-graphe induit* $G' = (V', E')$ de *G=(V,E)* est un graphe tel que $V' \subseteq V$ et $\forall (u,v) \in E$ si $u \in V'$ et $v \in V'$ alors $(u,v) \in E'$. Une *clique* $C = (V_c, E_c)$ d'un graphe G est un sous-graphe induit complet de G, tel que si $u \in V_c$ et si $v \in V_c$ alors $(u,v) \in E$.

2.2 Coloration et somme coloration

Une coloration du graphe est une fonction $c : V \mapsto \{1, 2, \ldots, k\}$ qui associe à chaque sommet $v \in V$ une couleur $c(v)$ représentée ici par un entier naturel. Une coloration est dite valide, si $\forall (u,v) \in E,\ c(u) \neq c(v)$. Nous notons $X = X_1, X_2, \ldots, X_k$ une coloration de G, avec $X_i = \{v \in V \mid c(v) = i\}$, nommée classe couleur i.

GCP a pour objectif de trouver une coloration valide du graphe, en utilisant un nombre minimal de couleurs. Ce nombre minimal est appelé *nombre chromatique* du graphe et ce nombre est appelé $\chi(G)$. Nous notons qu'une clique de n_c sommets requiert n_c couleurs.

Pour *MSCP*, à chaque couleur i est associé un poids p_i. A une coloration donnée X du graphe, correspond donc un poids $P(X)$ égal à la somme des poids des couleurs utilisées par X.

$$P(X) = p_1 \times |X_1| + p_2 \times |X_2| + \ldots + p_k \times |X_k|$$

MSCP consiste à trouver une coloration valide dont la somme des poids associés est minimale. Nous notons $\Sigma(G)$ cette somme, appelée *somme chromatique* du graphe.

$$\Sigma(G) = min\{P(X) \mid X \textit{ est une coloration valide de } G\}$$

Kubicka et Schwenk [14] ont démontré que *MSCP* est un problème *NP-difficile*. Le plus petit nombre de couleurs utilisées pour colorier le graphe G dans une solution optimale pour *MSCP* est appelé *force* du graphe et noté $s(G)$.

Comme les classes de couleur X_i forment des ensembles de sommets deux à deux indépendants, il est possible d'échanger deux couleurs c_i et c_j, sans attenter à la validité de la coloration. Nous noterons $X' = Exch_{(i,j)}(X)$ la coloration obtenue par un tel échange. Nous parlons alors de colorations symétriques (X' est symétrique à X). Néanmoins, les colorations correspondantes n'auront pas forcément un même poids associé. Nous définirons donc une relation de dominance entre les colorations afin d'éviter l'énumération de colorations symétriques. Sans perte de généralité nous pouvons considérer que : $\forall i, j \in \{1, 2, \ldots, k\}$ si $i < j$ alors $p_i < p_j$.

Définition 1 *Une coloration X' est dite dominée par une coloration X si $\exists\ c_i, c_j$, telles que $X' = Exch_{(i,j)}(X)$ et $P(X) \leq P(X')$.*

Propriété 1 *La coloration $X^\theta = X_1, X_2, \ldots, X_k$ pour laquelle est vérifiée $|X_1| \geq |X_2| \geq \ldots \geq |X_k|$ domine toutes les colorations X' symétriques à X^θ. X^θ est la coloration dominante de sa classe.*

Dans la suite de cette étude, comme dans la majorité des travaux de la littérature nous considèrerons que : $\forall i,\ p_i = i$. A la couleur c_i correspondra donc le poids i. Il est immédiat de trouver la coloration dominante X^θ à partir d'une coloration quelconque du graphe : il suffit d'ordonner les classes de couleurs suivant l'ordre décroissant de leur cardinalité. Donc, quelque soit la méthode employée pour trouver une coloration valide, nous nous rapportons toujours à la coloration X^θ lors de l'évaluation d'une solution.

2.3 Propriétés

Nous pouvons trouver dans la littérature quelques résultats théoriques et structurels relatifs à $MSCP$ [11, 3, 1, 21]. Celui qui nous intéresse plus particulièrement, fournit une borne supérieure et une borne inférieure théoriques, nous permettant de diminuer l'espace de recherche dans la résolution de $MSCP$. Alavi et *al* [1] ont ainsi démontré que pour un graphe composé de m arêtes, la somme chromatique est encadrée comme suit : $\lceil\sqrt{8m}\rceil \leq \Sigma(G) \leq \lfloor\frac{3(m+1)}{2}\rfloor$. De plus Malafiejski [21] montre que la force du graphe est bornée par le degré du graphe plus un : $s(G) \leq \Delta(G)+1$. Pour la suite, nous utilisons ces bornes pour l'initialisation de la borne inférieure et pour l'initialisation du nombre de couleurs à considérer dans le Branch-and-Bound et dans la formulation SAT et COP de $MSCP$. Nous notons que même si le nombre chromatique d'un graphe est connu, nous ne pouvons nous limiter à ce nombre dans l'exploration de l'espace des solutions. La figure 1 illustre bien cette particularité. En effet, pour cet arbre de 8 sommets et 7 arêtes, dont le nombre chromatique est trivialement 2, il est nécessaire d'utiliser 3 couleurs pour obtenir la somme chromatique. De manière générale, $s(G)$ peut être très éloigné de $\chi(G)$, et l'espace des solutions de $MSCP$ peut être plus compliqué à explorer que celui de GCP [13].

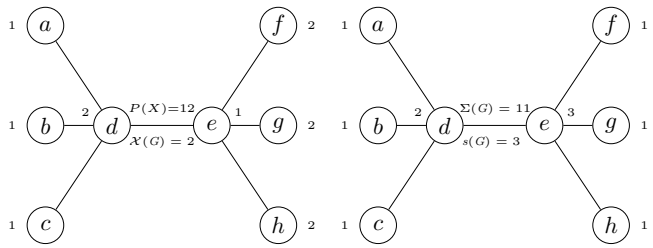


FIGURE 1 – Comparaison des solutions optimales pour GCP et $MSCP$

3 Branch-and-Bound pour MSCP

Nous présentons dans ce paragraphe une méthode de Branch-and-Bound (Séparation et Evaluation), pour la résolution exacte de $MSCP$. L'exploration complète de l'ensemble des solutions pourrait se représenter par un arbre de coloration comme le montre la figure 3 pour le cas d'une coloration du graphe de la figure 2. Chaque noeud de l'arbre correspond à un sommet du graphe et chaque arête pendante à une assignation de couleur pour le sommet correspondant. Le principe de la méthode développée, consiste à *séparer* l'ensemble des solutions en plusieurs sous-ensembles de solutions représentés par les différents noeuds, et à *évaluer* en chaque noeud la pertinence de continuer à explorer ce sous-ensemble de solutions. Les critères de séparation et évaluation employés sont décrits ci-dessous.

Séparation. Comme dans la majorité des méthodes énumératives, notre séparation de l'espace des solutions se fait par l'assignation d'une couleur à un sommet. Les couleurs sont comprises entre 1 et $\Delta(G) + 1$ comme mentionné dans le paragraphe précédent. Toute branche de longueur n, le nombre de sommets, correspond à une coloration X valide dont le poids associé est celui de la coloration dominante X^θ (cf. propriété 1). En chacun des noeuds, nous devons disposer de l'ensemble des couleurs *disponibles* pour le sommet correspondant, afin de séparer en autant de sous-ensembles. Pour plus de clarté dans la suite de l'exposé, nous convenons que la séparation relative au sommet v_i est effectuée au niveau i de notre arborescence (cf. figure 3).

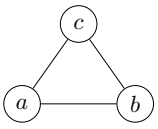


FIGURE 2 – Graphe

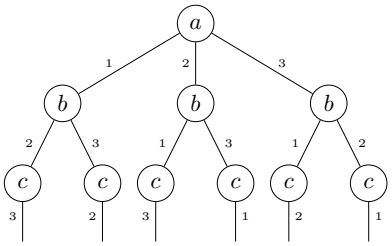


FIGURE 3 – Arbre de coloration pour $k = 3$ du graphe de la figure 2

Evaluation. Il nous est nécessaire pour cet évaluation de maintenir à jour une borne supérieure globale UB ainsi qu'une borne inférieure LB, relative au sous-ensemble de solutions représenté par le noeud en cours d'évaluation. Initialement ces bornes sont respectivement UB_{DSATUR} et $\lceil\sqrt{8m}\rceil$. $UB_{MDSATUR}$ est une borne supérieure calculée à partir de l'algorithme glouton $MDSATUR$ [18]. S'il s'avère lors de l'évaluation d'un noeud, que $LB \geq UB$, alors la branche est coupée, l'exploration de ce sous-ensemble de solutions est abandonnée, car toutes les solutions X contenues dans ce sous-ensemble verifient $P(X) \geq UB$.

Borne supérieure : Si une branche complète de l'arbre a pu être développée, elle correspond à une

coloration complète et valide X du graphe, et si $UB \geq P(X^\theta)$, alors la borne supérieure UB est mise à jour ($UB \leftarrow P(X^\theta)$).

Borne inférieure : La performance de cet algorithme repose également sur la qualité de la borne inférieure LB calculée en chaque noeud. Si nous considérons que nous nous trouvons au niveau i de l'arborescence, les sommets $v_1, v_2, \ldots, v_{i-1}$ ont une couleur qui leur a été préalablement assignée. Dans ce cas une borne inférieure triviale, LB^1, peut être déterminée en considérant par défaut que les $n-i+1$ sommets non coloriés, le sont avec leur plus petite couleur disponible, notée $ppcdispo(v_i)$. La valeur de cette borne est alors calculée comme suit :

$$LB^1 = \sum_{t=1}^{i-1} c(v_t) + \sum_{t=i}^{n} ppcdispo(v_t)$$

L'intérêt d'une telle borne est sa faible compléxité ($O(n)$), mais beaucoup trop de contraintes (les arêtes entre les sommets non coloriés) sont relâchées dans le calcul de LB^1 qui n'est donc pas très efficace dans la phase d'évaluation.

Nous proposons une borne inférieure préservant d'avantage les contraintes de voisinage du problème. Pour calculer cette borne, nous décomposons la partie du graphe non colorié noté $\widetilde{G} = (\widetilde{V}, \widetilde{E})$, en une partition de cliques $C_1, C_2, \ldots, C_l$ telle que $\forall i, j\ C_i \cap C_j = \emptyset$ et $\forall u \in \widetilde{G},\ \exists C_i\ tq.\ u \in C_i$. Pour obtenir une telle décomposition, nous utilisons un algorithme glouton $Clique_{Part}$ détaillé dans l'algorithme 3. Cet algorithme traite les sommets suivant un ordre fixe. Après avoir testé expérimentalement différents critères, nous avons choisi d'ordonnancer ces sommets suivant l'ordre croissant de leur degré. Chacune des cliques est alors coloriée en fonction des couleurs disponibles des sommets qui la composent. Nous notons $P_{lb}(C_i)$ la somme associée à la clique C_i. Considérons l'ensemble de toutes les couleurs disponibles d'une clique C_i : $Disp(C_i) = \bigcup_{v \in C_i} \{Dispo(v)\}$ et posons $|C_i| = \alpha_i$. Alors la somme des α_i plus petites couleurs disponibles ($P_{lb}(C_t)$) est une borne inférieure de la somme chromatique de la clique C_i. Par extension, la borne inférieure LB^2 est alors calculée comme suit :

$$LB^2 = \sum_{t=1}^{i-1} c(v_t) + \sum_{t=1}^{l} P_{lb}(C_t)$$

L'algorithme récursif $BBMSCP$ (cf. algorithme 1) synthétise l'ensemble de la méthode de séparation et évaluation présentée dans cet article. Dans cet algorithme, le paramètre G désigne le graphe non encore colorié, S la somme des couleurs assignées et UB la

Algorithme 1 : BBMSCP($G, \mathcal{S}, UB$)

```
Input  : G = (V, E)
         S somme des couleurs assignées
         UB borne supérieure
Output : Σ(G) la somme chromatique
1  begin
2    if | V |= ∅ then
3      return S;
4    Choisir v un sommet de G ;
5    Disp(v) := liste des couleurs disponibles de v
6    foreach c ∈ Disp(v) telle que S + c ⩽ UB
     do
7      foreach u ∈ N(v) do
8        Disp(u) := Disp(u) \ {c}
9      if LB²(G \ v) + S + c < UB then
10       UB :=BBMSCP(G \ v, S + c, UB);
11     foreach u ∈ N(v) do
12       Disp(u) := Disp(u) ∪ {c}
13   return UB;
```

Algorithme 2 : $LB^2(G)$

```
Input  : G = (V, E)
Output : lb une borne inférieure de Σ(G)
1  begin
2    lb ← 0;
3    Décomposer G en une partition de cliques
     C1, C2, ..., Cl;
4    foreach Ci do
5      Disp(Ci) := ⋃_{v∈Ci} Dispo(v)
6      foreach x := 1 to |Ci| do
7        c_min := MIN{c | c ∈ Disp(Ci)}
8        lb := lb + c_min
9        Disp(Ci) := Disp(Ci) \ {c_min}
10   return lb;
```

Algorithme 3 : $Clique_{Part}(G)$

Input : $G = (V, E)$
Output : P une partion en cliques de G

```
1  begin
2      P = ∅;
3      while G n'est pas vide do
4          v ← le plus petit sommet de G;
5          G ← G\{v};
6          if Il existe une clique C dans P tel que v
           est connecté à tous les sommets de C then
7              insérer v dans C;
8          else
9              créer une nouvelle clique C;
10             insérer v dans C;
11             P := P ∪ {C};
12     return P;
```

meilleure borne supérieure rencontrée. Le premier appel sera donc $BBMSCP(G, 0, UB_{MDSATUR})$. L'algorithme LB^2 (cf. algorithme 2) donne les détails du calcul de la borne inférieure proposée dans ce paragraphe, basée sur le partitionnement en cliques de la partie du graphe non encore colorié.

4 Résolution par codage de MSCP en SAT et COP

4.1 Préliminaire

4.1.1 SAT

Le problème *SAT* est un problème issu de la logique propositionnelle qui consiste à tester si une formule $\mathcal{F}$ sous forme normale conjonctive (*CNF*) possède une interprétation $\mathcal{I}$ satisfaisant toutes les clauses $\mathcal{F}$. Cependant, le problème *SAT* n'est pas suffisant pour modéliser un problème d'optimisation tel que le problème $MSCP$. En effet, il est nécessaire de considérer le problème *weighted partial MaxSAT* ou le problème dual *weighted partial MinSAT*. Le problème *weighted partial MaxSAT* est une extension du problème *MaxSAT*, respectivement, le problème *weighted partial MinSAT* est une extension du problème *MinSAT*. Ces deux problèmes sont des formules *CNF* constituées de clauses *dures* et de clauses souples (voir exemple 1). Dans le cas d'un problème *weighted partial MaxSAT*, le but est de trouver une affectation qui satisfasse toutes les clauses dures en maximisant la somme des poids des clauses souples satisfaites. Pour le cas du problème *weighted partial MinSAT*, l'objectif est de déterminer une affectation qui satisfasse toutes les clauses dures tout en minimisant la somme des poids des clauses souples satisfaites.

Exemple 1 *Soit une formule $\mathcal{F}$ bien formée pour le problème weighted partial MaxSAT composée des clauses $\{(C_1, \infty), (C_2, \infty), (C_3, 3), (C_4, 5)\}$, C_1=$x \vee y$,C_2=$\neg x \vee y$, C_3=x et $C_4 = \neg x$ où C_1 et C_2 sont des clauses dures, et C_3 et C_4 sont clauses souples. L'interprétation $\{x = faux, y = vrai\}$ est une solution optimale du problème weighted partial MaxSAT contrairement à l'interprétation $\{x = vrai, y = vrai\}$ qui est une solution optimale pour le problème weighted partial MinSAT.*

4.1.2 COP

Le problème de satisfaction de contraintes (*CSP*) a pour objectif de déterminer s'il existe une instanciation des variables telle que toutes les contraintes du problème soient satisfaites. Le formalisme simple de *CSP* permet de faciliter la modélisation de nombreux problèmes. Un *CSP* est défini par un triplet (X,D,C), où X est un ensemble fini de variables $\{x_1, x_2, ..., x_n\}$, D est l'ensemble des domaines associés à chaque variable du problème $\{dom(x_1), dom(x_2), ..., dom(x_n)\}$ et C est un ensemble fini de contraintes $\{c_1, c_2, ..., c_m\}$. Une contrainte $c_i \in C$ est une relation portant sur un ensemble de variables appartenant à X. Le problème d'optimisation sous contraintes (COP) est une extension du problème de satisfaction de contraintes (CSP) auquel est associé une fonction objectif F : $D_{x_1} \times D_{x_2} \times ... \times D_{x_n} \rightarrow \mathbb{N}$. Une solution optimale du CSP est un solution qui minimise ou maximise la fonction objectif tout en respectant l'ensemble de contraintes C. Un COP est donc défini par un quadruplet (X,D,C,F).

4.2 Codages de MSCP

4.2.1 vers SAT

À travers cette section, nous présentons le passage d'un problème *MSCP* vers un problème *weighted partial MaxSAT* et *weighted partial MinSAT*. Les variables booléennes x_{ai} du codage *SAT* sont définies comme suit :

$$x_{ai} = \begin{cases} 1 \textit{ si nous colorions le sommet } a \\ \textit{avec la couleur } i \\ \\ 0 \textit{ sinon} \end{cases}$$

Les clauses dures du problème *weighted partial MaxSAT* sont identiques à celles du problème *weighted partial MinSAT*. L'ensemble des clauses dures correspond

à l'ensemble des contraintes du problème *GCP*. Ces dernières sont les suivantes :

— Attribuer à un sommet quelconque une couleur $1, 2..., k$ (1) :

$$x_{a1} \vee x_{a2} \vee ... \vee x_{ak}$$

— Un sommet quelconque ne peut posséder qu'une seule couleur (2) :

$$\neg x_{ai} \vee \neg x_{aj} \textit{ pour tout } i \neq j$$

— Deux sommets voisins a et b ne peuvent pas être coloriés de la même façon (3) :

$$\neg x_{ai} \vee \neg x_{bi} \textit{ pour tout } i$$

Le nombre de clauses dures ainsi générées est :

$$\underbrace{n}_{(1)} + \underbrace{(n \times \binom{k}{2})}_{(2)} + \underbrace{(m \times k)}_{(3)}$$

L'ensemble des clauses souples est un ensemble de clauses unitaires correspondant aux sommets et aux couleurs. Chaque clause souple est pondérée selon un poids. Dans le cas du problème *weighted partial MaxSAT*, nous souhaitons maximiser la somme des poids des clauses souples satisfaites, contrairement au problème *weighted partial MinSAT* qui a pour objectif de minimiser la somme des poids des clauses souples satisfaites. Dans les deux cas, le nombre de clauses souples générées est de :

$$(n \times k)$$

La réécriture d'une instance $MSCP$ génère une CNF de complexité spatiale en $\mathcal{O}(n \times k^2 + m \times k)$.

Exemple 2 *Si nous prenons le graphe de la figure 1, pour $k = 3$, l'instance générée est la suivante :*

clauses dures :
$x_{a1} \vee x_{a2} \vee x_{a3}$
$x_{b1} \vee x_{b2} \vee x_{b3}$
$x_{c1} \vee x_{c2} \vee x_{c3}$
$\neg x_{a1} \vee \neg x_{a2}$
$\neg x_{a1} \vee \neg x_{a3}$
$\neg x_{a2} \vee \neg x_{a3}$
$\neg x_{b1} \vee \neg x_{b2}$
$\neg x_{b1} \vee \neg x_{b3}$
$\neg x_{b2} \vee \neg x_{b3}$
$\neg x_{c1} \vee \neg x_{c2}$
$\neg x_{c1} \vee \neg x_{c3}$
$\neg x_{c2} \vee \neg x_{c3}$
$\neg x_{a1} \vee \neg x_{b1}$
$\neg x_{a2} \vee \neg x_{b2}$
$\neg x_{a3} \vee \neg x_{b3}$
$\neg x_{a1} \vee \neg x_{c1}$
$\neg x_{a2} \vee \neg x_{c2}$
$\neg x_{a3} \vee \neg x_{c3}$
$\neg x_{b1} \vee \neg x_{c1}$
$\neg x_{b2} \vee \neg x_{c2}$
$\neg x_{b3} \vee \neg x_{c3}$

weighted partial MaxSAT (encodage 1)

clauses souples :
x_{a1} 3
x_{a2} 2
x_{a3} 1
x_{b1} 3
x_{b2} 2
x_{b3} 1
x_{c1} 3
x_{c2} 2
x_{c3} 1

weighted partial MinSAT (encodage 1)

clause souples :
$\neg x_{a1}$ 3
$\neg x_{a2}$ 2
$\neg x_{a3}$ 1
$\neg x_{b1}$ 3
$\neg x_{b2}$ 2
$\neg x_{b3}$ 1
$\neg x_{c1}$ 3
$\neg x_{c2}$ 2
$\neg x_{c3}$ 1

Dans l'instance MaxSAT et l'instance MinSAT, un poids plus grand est associé à une clause souple correspondant à une couleur plus petite, pour que le solveur MaxSAT (MinSAT) la satisfasse (falsifie) en priorité. En effet, la satisfaction (falsification) d'une clause souple dans MaxSAT (MinSAT) signifie l'affectation de la couleur correspondante au sommet correspondant. Par exemple, la satisfaction de la clause

x_{11} dans MaxSAT signifie l'affectation de la couleur 1 au sommet 1, de même que la la falsification de la clause $\neg x_{11}$ dans MinSAT. À noter que la seule différence entre le codage MinSAT et le codage MaxSAT est le signe des clauses souples. Comme nous allons voir, cette petite différence a un très grand impact dans la résolution de MSCP. En effet, la résolution MinSAT repose sur le calcul d'une borne supérieure détectant les clauses souples ne pouvant être falsifiées en même temps. Avec cet encodage, nous observons par exemple que les clauses souples $\neg x_{a1}$ et $\neg x_{a2}$ ne peuvent être falsifiées au même moment car la clause dure $\neg x_{a1} \vee \neg x_{a2}$ serait violée. À travers cet exemple nous pouvons donc observer que le codage MinSAT permet une meilleure prise en considération des spécificités de l'instance $MSCP$. Un autre encodage valide $MinSAT$ et $MaxSAT$ est d'utiliser le même ensemble de clauses dures et de coder l'ensemble des clauses souples de la façon suivante :

Exemple 3

weighted partial MaxSAT (encodage 2)

$$\begin{array}{rl} \textit{clause souples}: & \neg x_{a1}\ 1 \\ & \neg x_{a2}\ 2 \\ & \neg x_{a3}\ 3 \\ & \neg x_{b1}\ 1 \\ & \neg x_{b2}\ 2 \\ & \neg x_{b3}\ 3 \\ & \neg x_{c1}\ 1 \\ & \neg x_{c2}\ 2 \\ & \neg x_{c3}\ 3 \end{array}$$

weighted partial MinSAT (encodage 2)

$$\begin{array}{rl} \textit{clause souples}: & x_{a1}\ 1 \\ & x_{a2}\ 2 \\ & x_{a3}\ 3 \\ & x_{b1}\ 1 \\ & x_{b2}\ 2 \\ & x_{b3}\ 3 \\ & x_{c1}\ 1 \\ & x_{c2}\ 2 \\ & x_{c3}\ 3 \end{array}$$

Il apparaît clairement que ce codage n'est pas du tout adapté au problème MinSAT. En effet, contrairement au codage précédent il est seulement possible d'en déduire que seules les clauses souples x_{a1}, x_{a2} et x_{a3} ne peuvent être falsifiées simultanément. Le calcul de la borne supérieure s'en trouve dégradé. Il est à noter que pour le problème MaxSAT, le choix de l'encodage 2 est plus efficace. Une raison est le fait de la proximité de la lower bound et l'upper bound durant la résolution de MaxSAT [15]. Dans la suite de ce papier, l'encodage 1 est utilisé pour la transformation de $MSCP$ en un problème $MinSAT$ et l'encodage 2 est utilisé pour la transformation de $MSCP$ en un problème $MaxSAT$.

4.2.2 vers COP

Pour transformer un problème de somme coloration en COP il est dans un premier temps nécessaire de déterminer l'ensemble des variables de X, puis l'ensemble des domaines de chaque variable, et enfin déterminer les différentes contraintes liées au problème $MSCP$. Nous définissons donc le COP de la façon suivante :

- X=$\{v_1, v_2, ..., v_n\}$ chaque variable v_i correspond à un sommet du graphe ;
- $dom(v_1)$=$dom(v_2)$=...=$dom(v_n)$=$\{1,2,...,k\}$ correspond au différentes couleurs possibles ;
- Si v_i et v_j sont deux sommets du graphe tel que $(v_i, v_j) \in E$ alors $c_{ij} \in C$ et nous définissons c_{ij} : $v_i \neq v_j$;
- La fonction objectif à minimiser : $F = \sum_{i=1}^{n} v_i$.

La réécriture d'une instance $MSCP$ génère un CSP de complexité spatiale en $\mathcal{O}(n+m)$.

Exemple 4 *Si nous prenons le graphe de la figure 1, pour un k valant 3, le COP (X,D,C) générée est la suivante :*

- *X=*$\{a,b,c\}$ *;*
- *dom(a)=dom(b)=dom(c)=*$\{1,2,3\}$ *;*
- *C=*$\{c_{ab}, c_{ac}, c_{bc}\}$ *;*
- *minimiser* $F = a + b + c$.

5 Resultat

Nous avons mené nos expérimentations sur un processeur Intel Westmere Xeon E7-8837 de 2.66GHz. Comme il s'agit du tout début de nos travaux, les seules instances que nous avons utilisées sont des graphes aléatoires et quelques instances issues de COLOR [10] et DIMACS [6].

Les graphes aléatoires sont des instances de taille et de densité variables. Un graphe aléatoire est construit de la façon suivante : la densité d d'un graphe aléatoire est comprise entre 0.1 et 0.9. Pour chaque couple u, v de sommets, l'arête (u, v) est ajoutée au graphe avec la probabilité d. Les graphes structurés ont été utilisés comme Benchmark lors du "computational symposium" [10] et lors du deuxième challenge DIMACS [6]. Ils correspondent à des applications modélisés par des graphes.

5.1 Résultats expérimentaux

Nous comparons les méthodes suivantes :

— BBMSCP : l'algorithme de Branch-and-Bound que nous proposons dans ce papier ;
— MaxSatz [16] : nous avons utilisé la version de ce solveur de *weighted partial MaxSAT* qui a participé à l'évaluation de MaxSAT de 2009 ;
— ISAC [2] : nous avons utilisé la version de ce solveur de *weighted partial MaxSAT* qui a participé à l'évaluation de MaxSAT de 2013 ;
— MinSatz [17] : nous avons utilisé la version *weighted partial MinSAT* de 2013 ;
— Choco3 [9] : nous avons utilisé la version 3.3.0.

Le tableau 1 synthétise l'ensemble des résultats expérimentaux obtenus pour un ensemble d'instances aléatoires, tandis que le tableau 2 présente les résultats pour les instances structurées. Sont représentés dans les différentes colonnes, N le nombre de sommets, M le nombre d'arêtes, $\Sigma(G)$ la somme chromatique et pour *BBMSCP*, *MaxSatz*, *ISAC*, *MinSatz* et *Choco3* les temps d'exécution de chaque solveur en secondes. Si après 1 heure d'exécution la methode n'est pas terminée, le processus est stoppé (N/A).

Ces résultats préliminaires montrent l'efficacité des solveurs *weighted partial MinSAT* et *weighted partial MaxSAT* dans la résolution du problème $MSCP$, notamment le solveur ISAC. Une première explication est que pour couper les branches, ces solveurs utilise une borne calculée en exploitant des raisonnements propositionnels comme les règles de référence et la propagation unitaire en prenant en compte des relations entre les clauses souples. Pour permettre une exploitation optimale de la borne supérieure de $MinSAT$ et la borne inférieure de $MaxSAT$, il est nécessaire d'adapter le codage de l'instance $MinSAT$ et $MaxSAT$. Cette remarque est confirmée par le tableau 3. Celui-ci contient les résultats expérimentaux obtenus de ces deux encodages (cf exemple 2 et 3) pour un ensemble d'instances aléatoires en utilisant les solveurs $MinSatz$, $MaxSatz$ et $ISAC$.

Nous pouvons observer que les résultats confirment l'importance majeure du choix de l'encodage pour la résolution de $MaxSAT$ et $MinSAT$. Ces résultats nous montrent que l'efficacité de la résolution d'un problème ne dépend pas uniquement du solveur, mais également du choix de l'encodage du problème. Nous notons que le solveur MinSatz n'est pas adapté pour la résolution de problèmes de grande taille, car le nombre maximum de clauses souple est limité à 10000. Ceci est dû à l'utilisation d'une matrice adjacente implémentée dans le solveur MinSatz. Nous allons remédier à ce problème dans la prochaine amélioration de MinSatz.

Malgré le peu de différence entre le codage d'une instance $MaxSAT$ et d'une instance $MinSAT$, nous observons que le problème $MinSAT$ peut être pertinent dans la résolution de certains problèmes. Nous notons que la complexité spatiale du codage de $MSCP$ en CNF est quadratic en nombre de couleurs, et que cette complexité est linéaire pour le codage de $MCSP$ en COP. Cependant, les tableaux 1 et 2 montrent des résultats peu encourageant pour la résolution de $MSPC$ par COP.

Nous pouvons également voir que le solveur BBMSCP, première ébauche de nos travaux, reste compétitif. En effet, le nombre d'instances résolues par le solveur BBMSCP est supérieur à celui des instances résolues par Choco3, et il possède un temps de résolution comparable aux solveurs MaxSatz, ISAC et MinSatz.

6 Conclusion

À travers cet article, nous avons présenté *MSCP* et proposé différentes méthodes exactes de résolution. La première méthode se base sur un algorithme de branch-and-bound ($BBMSCP$), pour lequel les premiers résultats montrés sont prometteurs. $BBMSCP$ est à notre connaissance le premier solveur exact dédié à la résolution de $MSCP$. Les résultats obtenus nous encouragent à développer de nouvelles bornes et de nouveaux types de branchement pour BBMSCP, afin d'en augmenter l'efficacité sur les instances de plus grande taille. Nous avons également proposé un codage sous forme de problème *weighted partial MinSAT* et *weighted partial MaxSAT*, et remarqué qu'ils sont particulièrement adapté pour la résolution de *MSCP*. Une piste très prometteuse serait de combiner l'exploitation des structures des graphes et des bornes de $BBMSCP$ avec la puissance de raisonnement logique de ces solveurs dans la résolution de $MSCP$.

Remerciements : Nous remercions sincèrement les reviewers pour leurs remarques constructives qui nous ont permis d'améliorer la présentation de nos travaux.

				BBMCP	MaxSatz	ISAC	MinSatz	Choco3
Instances	N	M	$\Sigma(G)$	Temps	Temps	Temps	Temps	Temps
S10A23	10	23	20	0	0	0	0	30
S10A40	10	40	34	0	0	0	0	48
S10A5	10	5	14	0	0	0	0	3
S20A100	20	100	55	10	126	0	172	N/A
S20A19	20	19	27	0	0	0	0	2
S20A50	20	50	39	0	0	0	0	74
S20A75	20	75	52	8	144	0	18	N/A
S20A95	20	95	50	3	29	0	32	307
S25A100	25	100	63	96	1051	0	150	N/A
S25A30	25	30	40	0	0	0	0	25
S30A100	30	100	65	105	242	0	22	N/A
S30A44	30	44	48	0	0	0	0	1551
S35A60	35	60	60	54	2	0	0	N/A

TABLE 1 – Comparaison du temps d'exécution en secondes nécessaire à la détermination de la somme chromatique d'un pool d'instances aléatoires.

				BBMCP	MaxSatz	ISAC	MinSatz	Choco3
Instances	N	M	$\Sigma(G)$	Temps	Temps	Temps	Temps	Temps
1FI3	30	100	54	2	9	0	2	707
miles250.col	128	387	334	N/A	2	0	0	N/A
miles500.col	128	1170	715	N/A	20	0	0	N/A
myciel3.col	11	20	21	0	0	0	0	3
myciel4.col	23	71	45	0	9	0	0	1944
queen5_5.col	25	160	75	3036	N/A	N/A	N/A	N/A

TABLE 2 – Comparaison du temps d'exécution en secondes nécessaire à la détermination de la somme chromatique d'un pool d'instances structurées.

Références

[1] Y. Alav, P.J. Malde, and A.J. Schwenk. Tight bounds on the chromatic sum of a connected graph. *Journal of Graph Theory*, 13 :353–357, 1989.

[2] C. Ansótegui, Y. Malitsky, and M. Sellmann. Maxsat by improved instance-specific algorithm configuration. *Association for the Advancement of Artificial Intelligence*.

[3] A. Bar-Noy and G. Kortsarz. Minimum color sum of bipartite graphs. *J. Algolrithms*, 28(2) :339–365, 1998.

[4] U. Benlic and J.K. Hao. A study of breakout local search for the minimum sum coloring problem. *SEAL 2012, Lecture Notes in Computer Science*, 7673 :128–137, 2012.

[5] D. Brélaz. New methods to color vertices of a graph. *Commun. ACM*, 22(4) :251–256, 1979.

[6] ftp ://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/.

[7] R. Garey and S. Johnson. *Computers and Intractability ; A Guide to the Theory of NP-Completeness*. 1990.

[8] A. Hertz, M. Plumettaz, and N. Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13) :2551 – 2560, 2008.

[9] http ://choco solver.org.

[10] http ://mat.gsia.cmu.edu/COLOR/instances.html.

[11] K. Jansen. The optimum cost chromatic partition problem. In *CIAC*, pages 25–36, 1997.

[12] Y. Jin, J.K. Hao, and J.P. Hamiez. A memetic algorithm for the minimum sum coloring problem. *Computers & OR*, 43 :318–327, 2014.

[13] L.G. Kroon, A. Sen, H. Deng, and A. Roy. The optimal cost chromatic partition problem for trees and interval graphs. In *WG'96*, pages 279–292, 1997.

				MinSatz		MaxSatz		ISAC	
				Encodage 1	Encodage 2	Encodage 1	Encodage 2	Encodage 1	Encodage 2
Instances	N	M	$\Sigma(G)$	Temps	Temps	Temps	Temps	Temps	Temps
S10A23	10	23	20	0	0	0	0	0	0
S10A40	10	40	34	0	15	0	0	0	0
S10A5	10	5	14	0	0	0	0	0	0
S20A100	20	100	55	172	N/A	N/A	126	N/A	0
S20A19	20	19	27	0	2213	0	0	0	0
S20A50	20	50	39	0	N/A	8	0	0	0
S20A75	20	75	52	18	N/A	1372	144	N/A	0
S20A95	20	95	50	32	N/A	N/A	29	N/A	0
S25A100	25	100	63	150	N/A	N/A	1051	N/A	0
S25A30	25	30	40	0	N/A	0	0	0	0
S30A100	30	100	65	22	N/A	N/A	242	N/A	0
S30A44	30	44	48	0	N/A	3098	0	N/A	0
S35A60	35	60	60	0	N/A	N/A	2	N/A	0

TABLE 3 – Comparaison du temps d'exécution en secondes nécessaire à la détermination de la somme chromatique d'un pool d'instances aléatoires entre l'encodage du problème MinSAT 1 et 2.

[14] E. Kubicka and A.J. Schwenk. An introduction to chromatic sums. In *CSC'89 : Proceedings of the 17th conference on ACM Annual Computer Science Conference*, pages 39–45, New York, NY, USA, 1989. ACM.

[15] C.M. LI, F. Manya, and J. Planes. Exploiting unit propagation to compute lower bounds in branch and bound maxsat solvers. In *CP 05 : Proceedings of the 11st international conference on Principles and Practice of Constraint Programming*, pages 403–414, Sitges, SPAIN, 2005.

[16] C.M. Li, F. Manyà, and J. Planes. New inference rules for max-sat. *Journal Of Arteficial Intetelligence Research*, 30 :321–359, 2007.

[17] C.M. Li, Z. Zhu, F. Manyà, and L. Simon. Minimum satisfiability and its applications. *IJCAI*, 2011.

[18] Y. Li, C. Lucet, A. Moukrim, and K. Sghiouer. Greedy algorithms for the minimum sum coloring problem. In *International Workshop : Logistics and transport*, 2009.

[19] Z. Lü and J.K. Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1) :241 – 250, 2010.

[20] C. Lucet, F. Mendes, and A. Moukrim. An exact method for graph coloring. *Computers & Operations Research*, 33(8) :2189–2207, 2006.

[21] M. Malafiejski. *Sum coloring of graphs*, chapter 4, pages 55–66. Graph Coloring. New-York (USA), 2004.

[22] I. Méndez-Díaz and P. Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5) :826–847, 2006.

[23] A. Moukrim, K. Sghiouer, C. Lucet, and Y. Li. Lower bounds for the minimum sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36 :663–670, 2010.

[24] D.C. Porumbel, J.K. Hao, and P. Kuntz. A search space "cartography" for guiding graph coloring heuristics. *Comput. Oper. Res.*, 37 :769–778, April 2010.

[25] K. Sghiouer, Y. Li, C. Lucet, and A. Moukrim. A memetic algorithm for the minimum sum coloring problem. In *Conférence Internationale en Recherche Opérationnelle*, 2010.

[26] Z. Zhou, C.M. Li, C.H. Huang, and R. Xu. An exact algorithm with learning for the graph coloring problem. *Computers & OR*, 51 :282–301, 2014.

La Contrainte Smart Table

Jean-Baptiste Mairy[1] **Yves Deville**[1] **Christophe Lecoutre**[2]

[1] ICTEAM, Université catholique de Louvain, Belgique
[2] CRIL-CNRS UMR 8188, Université d'Artois, F-62307 Lens, France
{jean-baptiste.mairy,yves.deville}@uclouvain.be lecoutre@cril.fr

Résumé

Cet article est un résumé en français de [5]. Les contraintes table sont très utiles pour la modélisation de problèmes combinatoires complexes en Programmation par Contraintes (PPC). Elles sont un moyen universel de représentation, mais malheureusement, la taille des tables peut grandir de manière exponentielle. Dans [5], nous proposons la possibilité d'introduire des contraintes arithmétiques simples au niveau des entrées des tables. Les tuples classiques de valeurs sont, dès lors, remplacés par des smart tuples. Il est possible de voir les contraintes smart table comme des combinaisons logiques de contraintes arithmétiques simples. Cette nouvelle forme de tuples permet un encodage compact de nombreuses contraintes, incluant une douzaine de contraintes globales connues. Dans [5], nous montrons également comment, avec la seule exigence d'acyclicité des smart tuples, un algorithme de filtrage de cohérence d'arc (généralisée) peut être conçu. Les expériences réalisées montrent que la contrainte table smart est un outil générique prometteur pour la PPC.

Les contraintes table donnent explicitement la liste des tuples acceptés (ou interdits). Cette liste est appelée la table de la contrainte. Ces contraintes peuvent théoriquement encoder n'importe quelle contrainte (d'une certaine manière, on peut les qualifier de contraintes universelles). Malheureusement, la taille des tables peut augmenter exponentiellement avec l'arité des contraintes, ce qui représente un problème de taille. Parmi les solutions proposées à ce problème, deux d'entre elles correspondent à une modification de la définition des tuples. Il s'agit des tuples compressés [3, 6, 8] et des supports courts appliqués aux contraintes table [2]. Les tuples compressés permettent de remplacer certaines valeurs par des ensembles ; un tuple compressé représente alors le produit cartésien des différents ensembles impliqués. Les supports courts appliqués aux contraintes table permettent de ne spécifier aucune valeur pour certaines variables au niveau d'un tuple ; une variable non spécifiée pouvant prendre n'importe quelle valeur de son domaine. Dans [5], nous proposons de généraliser ces deux approches en introduisant, au niveau des entrées de tables, des contraintes arithmétiques simples. Ces tuples sont appelés *smart tuples* et les contraintes table sur des smart tuples, des contraintes *smart table*. Par exemple, l'ensemble des tuples $\{(1,2,1), (1,3,1), (2,2,2), (2,3,2), (3,2,3), (3,3,3)\}$ pour les variables $\{x_1, x_2, x_3\}$ dont les domaines sont $\{1,2,3\}$ peut être représenté par le smart tuple suivant :

x_1	x_2	x_3
$= x_3$	≥ 2	$*$

ou, sous une forme équivalente par $(x_1 = x_3, x_2 \geq 2)$. Une étoile dans la forme tabulée signifie que la variable peut prendre n'importe quelle valeur de son domaine, si elle n'est pas contrainte ailleurs (ce qui n'est pas le cas dans l'exemple).

Il est à noter que les smart tuples peuvent être vus comme une disjonction de conjonctions de contraintes arithmétiques simples. Les combinaisons logiques de contraintes ont été largement étudiées dans la littérature. L'originalité de l'approche présentée dans ce papier est que la forme des smart tuples (disjonctions de conjonctions, conjonctions formant des réseaux acycliques) permet de définir un algorithme de filtrage efficace pour obtenir la cohérence d'arc généralisée (GAC). Les contraintes smart table peuvent être perçues comme un sous ensemble de l'algèbre logique définie dans [1]. Les règles de filtrage pour contraintes smart table sont d'ailleurs directement dérivées de celles présentées dans [1]. La restriction aux disjonctions de conjonctions est motivée par un désir de garder la sémantique des contraintes smart table proche de celle des contraintes table classiques. La restriction aux conjonctions acycliques est une nécessité pour pouvoir utiliser les règles de filtrage de [1] et garantir l'obtention de GAC de manière efficace.

1 Définition de la contrainte smart table

Une contrainte smart table est définie par un ensemble de smart tuples. Cet ensemble de smart tuples est appelé la smart table et représenté par $table(sc)$ pour une contrainte smart table sc. Un smart tuple σ est lui même un ensemble de contraintes de tuple. Une contrainte tuple peut prendre quatre formes différentes :

1. <var> <op> a
2. <var> $\in S$ ou <var> $\notin S$
3. <var> <op> <var>
4. <var> <op> <var> + b

où <var> est une variable de la portée de la contrainte smart table, a et b, des constantes, S, un ensemble de constantes et <op> un opérateur choisi dans $\{<, \leq, =, \neq, \geq, >\}$. La sémantique d'une contrainte smart table est simple et naturelle : un tuple classique τ est accepté par une contrainte smart table sc ssi il existe $\sigma \in table(sc)$ tel que τ satisfait σ. Une variable de la contrainte qui n'est pas contrainte par une contrainte de tuple peut prendre n'importe quelle valeur de son domaine dans le smart tuple.

Les contraintes smart table peuvent être utilisées pour modéliser efficacement de nombreuses contraintes, y compris certaines contraintes globales bien connues. Nous donnons ci-dessous un exemple de contrainte smart table encodant la contrainte Element. Les contraintes de tuple sont écrites directement dans la table de la contrainte dans la colonne correspondant à la première variable de la contrainte de tuple. Par exemple, la contrainte de tuple $x_1 \leq x_3 + 2$ sera représentée par $\leq x_3 + 2$ dans la colonne de x_1.

Element$(I, [x_1, x_2, \ldots, x_m], R)$ *:* $\bar{\boldsymbol{x}}[I] = R$

I	x_1	$\ldots$	x_m	R
$= 1$	$*$	$\ldots$	$*$	$= x_1$
$= 2$	$*$	$\ldots$	$*$	$= x_2$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$= m$	$*$	$\ldots$	$*$	$= x_m$

2 Filtrage de la contrainte smart table

Cette section concerne le filtrage GAC de la contrainte smart table. La propriété GAC nécessite que chaque littéral (couple variable-valeur) ait un support sur chaque contrainte. Un support pour un littéral est un tuple valide et accepté par la contrainte prouvant que le littéral peut satisfaire la contrainte avec les domaines courants. L'identification de l'ensemble des supports d'une contrainte permet donc en général d'obtenir GAC pour cette contrainte. Pour une contrainte smart table sc, chaque smart tuple σ correspond à un CSP P_σ. Les variables de P_σ sont les variables de la portée de sc et les contraintes sont les contraintes de tuples de σ. Nous imposons que les graphes de contraintes liés aux CSPs de chaque smart tuple soient acycliques. Les tuples classiques supportés par un smart tuple σ sont ceux dans $sol(P_\sigma)$. L'ensemble des supports pour une contrainte smart table est donc $\bigcup_{\sigma \in table(sc)} sols(P_\sigma)$. Calculer l'ensemble des solutions pour chaque CSP P_σ peut sembler coûteux. Heureusement, la forme acyclique des CSPs de smart tuples permet d'obtenir efficacement l'ensemble des littéraux apparaissant dans $sols(P_\sigma)$. En effet, nous avons la propriété suivante :

Propriété 1 *Si σ est un tuple smart d'une contrainte smart table et que P'_σ est la fermeture GAC de P_σ, alors P'_σ est globalement consistant.*

Cette propriété nous permet d'obtenir l'ensemble des supports en calculant la fermeture GAC des smart tuples.

L'algorithme de filtrage pour les contraintes smart table, que nous appelons smartSTR, est basé sur STR et STR2 [7, 4]. Le principe de fonctionnement de STR / STR2 est de parcourir les tuples de la table de la contrainte. La validité de chaque tuple est testée. Si le tuple est valide, les supports sont collectés. Sinon, le tuple est retiré de la table. Pour SmartSTR, un smart tuple σ est valide si P_σ est satisfaisable et les supports collectés sont ceux de $sols(P_\sigma)$. L'astuce de la fermeture GAC de P_σ est utilisée dans les deux cas.

Références

[1] Fahiem Bacchus and Toby Walsh. Propagating logical combinations of constraints. In *IJCAI*, pages 35–40, 2005.

[2] Christopher Jefferson and Peter Nightingale. Extending simple tabular reduction with short supports. In *Proceedings of IJCAI'13*, pages 573–579, 2013.

[3] George Katsirelos and Toby Walsh. A compression algorithm for large arity extensional constraints. In *Proceedings of CP'07*, pages 379–393, 2007.

[4] Christophe Lecoutre. STR2 : optimized simple tabular reduction for table constraints. *Constraints*, 16(4) :341–371, 2011.

[5] Jean-Baptiste Mairy, Yves Deville, and Christophe Lecoutre. The smart table constraint. In *Integration of AI and OR Techniques in Constraint Programming*. Springer International Publishing, 2015.

[6] Jean-Charles Régin. Improving the expressiveness of table constraints. In *Proceedings of CP'11 Workshop on Constraint Modelling and Reformulation*, 2011.

[7] Julian R Ullmann. Partition search for non-binary constraint satisfaction. *Information Sciences*, 177(18) :3639–3678, 2007.

[8] Wei Xia and Roland H. C. Yap. Optimizing STR algorithms with tuple compression. In *Proceedings of CP'13*, pages 724–732, 2013.

Recherche d'un plus grand sous-graphe commun par décomposition du graphe de compatibilité

Maël Minot[1,2] **Samba Ndojh NDIAYE**[1,3] **Christine SOLNON**[1,2]

[1] Université de Lyon - LIRIS
[2] INSA-Lyon, LIRIS, UMR5205, F-69621, France
[3] Université Lyon 1, LIRIS, UMR5205, F-69622 France

{mael.minot,samba-ndojh.ndiaye,christine.solnon}@liris.cnrs.fr

Résumé

La taille d'un plus grand sous-graphe commun permet de mesurer la similarité entre des objets représentés par des graphes. Cependant, trouver un tel sous-graphe est un problème $\mathcal{NP}$-difficile. Nous décrivons dans cet article une méthode de décomposition, basée sur le graphe de compatibilité, qui produit des sous-problèmes indépendants. Des résultats expérimentaux montrent l'efficacité de cette méthode sur des instances difficiles.

Abstract

The size of a maximum common induced subgraph serves as a measure to evaluate the similarity level between objects represented by graphs. However, finding such a subgraph is an $\mathcal{NP}$-hard problem. In this article, we describe a decomposition method, based on the compatibility graph, that produces independent subproblems. Experiments show that this method obtains very good results on difficult instances.

1 Introduction

La recherche d'un plus grand sous-graphe commun est un problème $\mathcal{NP}$-difficile particulièrement complexe qui n'en reste cependant pas moins intéressant du fait de ses nombreuses applications en chimie, biologie ou encore en traitement d'images, lorsqu'il est nécessaire de mesurer la similarité d'objets représentés par des graphes. Il existe deux principaux types d'approches complètes permettant de résoudre ce problème. La technique dite du *branch and bound* construit incrémentalement tous les sous-graphes communs possibles [16]. Une construction de sous-graphe est alors stoppée dès que l'algorithme parvient à prouver que ce sous-graphe ne permettra pas de construire une solution plus grande que la meilleure trouvée jusqu'alors. Les améliorations de cette technique impliquent généralement de trouver des moyens d'éliminer des branches plus tôt en prouvant leur inutilité. La Programmation par Contraintes (PPC, ou CP en anglais), notamment, permet de réduire le nombre de branches en propageant des contraintes [18, 26]. La PPC est compétitive avec l'état de l'art pour résoudre ce problème, mais il reste difficile de résoudre des instances dont les graphes ont plus de quelques dizaines de sommets. La seconde approche complète pour la recherche d'un plus grand sous-graphe commun est basée sur une reformulation du problème en un graphe dit *de compatibilité* [1, 2, 8, 13]. Chaque clique présente dans ce graphe de compatibilité correspond à un sous-graphe commun, et une clique *maximum* est un sous-graphe commun *maximum*. Ainsi, il devient possible de se contenter de résoudre le problème de la clique maximum dans le graphe de compatibilité.

Afin d'améliorer le processus de résolution, le problème peut être décomposé en sous-problèmes indépendants, qui offrent donc la possibilité d'être résolus en parallèle. Une telle approche a été présentée par exemple dans [7, 14, 15] pour le problème de la clique maximum, ou dans [21, 22] pour les problèmes de satisfaction de contraintes. Dans la plupart des cas, cette décomposition prend la forme d'un découpage des domaines. Par exemple, Régin et al. proposent dans [21, 22] de créer p sous-problèmes indépendants en choisissant k variables telles que la taille du produit cartésien de leurs domaines est proche de p. Un sous-

problème est alors généré pour chaque instanciation possible de ces k variables, excepté pour les instanciations provoquant d'emblée une inconsistance vis-à-vis des contraintes du problème (« Non Detected Inconsistent subproblems »), étant donné qu'il s'agit de branches qui n'auraient pas été considérées par une recherche séquentielle.

Un autre moyen de décomposer un problème est de tirer parti de sa structure [11, 5, 6]. Ces méthodes impliquent généralement une décomposition arborescente du graphe de contraintes. Cependant, dans le cas du problème du plus grand sous-graphe commun, les associations de sommets doivent être strictement distinctes, ce qui induit l'utilisation de contraintes *allDifferent* dans les modèles de PPC. La portée de ces contraintes est égale à l'ensemble des variables dans son entier, ce qui rend inutile toute méthode basée sur une décomposition arborescente du fait de l'apparition d'un sous-problème équivalent au problème initial, non-décomposé. Une variante proposée par [3] se base sur la microstructure du problème de satisfaction de contraintes plutôt que sur son graphe de contraintes. Cette nouvelle approche produit des problèmes complètement indépendants, ce qui permet de conserver la possibilité de les résoudre en parallèle.

Dans cet article, nous appliquons cette technique au problème du plus grand sous-graphe commun, afin de le décomposer en un ensemble de problèmes indépendants tout en réduisant la taille de l'espace de recherche à explorer. Ces travaux sont la continuation de ceux présentés dans [17].

La section qui suit présente le problème du plus grand sous-graphe commun de manière plus détaillée, ainsi que quelques notions de PPC. Une méthode de décomposition destinée aux problèmes de PPC sera également résumée. La section 3 explique comment adapter cette méthode de décomposition au problème du plus grand sous-graphe commun. La section 4 donne les premiers résultats expérimentaux et les analyse, introduisant ainsi la section 5, qui propose des voies d'amélioration des performances. Les résultats finaux ainsi obtenus sont quant à eux présentés en section 6.

2 Préliminaires

2.1 Plus grand sous-graphe commun

Définition 1 *Un graphe* non orienté G *est défini par un ensemble fini de nœuds* N_G *et un ensemble* $E_G \subseteq N_G \times N_G$ *d'arêtes. Chaque arête est un couple non orienté de nœuds.*
Dans un graphe orienté, E_G *est un ensemble d'*arcs, *qui sont des couples* orientés *de nœuds.*

Dans les définitions suivantes, nous nous plaçons dans le cadre des graphes *orientés*. Cependant, les notions abordées peuvent être étendues aux graphes non orientés.

Définition 2 *Soient deux graphes* G *et* G'. G *est* isomorphe *à* G' *s'il existe une bijection* $f : N_G \rightarrow N_{G'}$ *préservant les arêtes,* i.e., $\forall(u,v) \in N_G \times N_G, (u,v) \in E_G \Leftrightarrow (f(u),f(v)) \in E_{G'}$.

Définition 3 G' *est un sous-graphe induit de* G *si* $N_{G'} \subseteq N_G$ *et* $E_{G'} = E_G \cap (N_{G'} \times N_{G'})$. *On dit alors que* G' *est le sous-graphe de* G *induit par le sous-ensemble de nœuds* $N_{G'}$, *et on note* $G' = G_{\downarrow N_{G'}}$.

En d'autres termes, G' est obtenu à partir de G en retirant tout nœud de G qui ne se trouve pas dans $N_{G'}$, et en conservant uniquement les arêtes dont *les deux extrémités* sont présentes dans $N_{G'}$.

Les sous-graphe dits *partiels* sont, eux, obtenus en considérant un sous-ensemble $N_{G'}$ de N_G et un sous-ensemble d'arêtes de G dont les extrémités sont toutes deux dans $N_{G'}$, comme formalisé dans la définition suivante :

Définition 4 G' *est un sous-graphe* partiel *de* G *si* $N_{G'} \subseteq N_G$ *et* $E_{G'} \subseteq E_G \cap (N_{G'} \times N_{G'})$. *Un sous-graphe partiel* G' *de* G *induit par un sous-ensemble* $E_{G'}$ *de* E_G *est défini par* $(N_{G'},E_{G'})$, *où* $N_{G'} = \cup_{(x,y)\in E_{G'}}(x,y)$.

Définition 5 *Un* sous-graphe commun *de deux graphes* G *et* G' *est un graphe qui est isomorphe à des sous-graphes de* G *et* G'.

Définition 6 *Un plus grand sous-graphe induit commun (ou « MCIS » pour* « Maximum Common Induced Subgraphe »*) est un sous-graphe induit commun avec un nombre de nœuds maximal.*

Définition 7 *Un plus grand sous-graphe partiel commun (ou « MCPS » pour* « Maximum Common Partial Subgraphe »*) est un sous-graphe partiel commun avec un nombre d'arêtes maximal.*

La figure 1 expose des exemples de MCIS et de MCPS.

Dans cet article, nous nous restreignons à la recherche d'un MCIS. Cependant, les résultats peuvent aisément être étendus au problème du MCPS, comme le montre [18].

2.2 Modèle de PPC pour le problème du MCIS

Définition 8 *Un problème de satisfaction de contraintes (« CSP » pour* « Constraint Satisfaction Problem »*) est défini par un triplet* (X, D, C) *: X est*

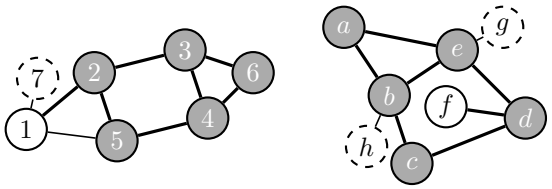


FIGURE 1 – Les nœuds pleins de ces deux graphes forment un MCIS. Une bijection peut être définie de la manière suivante : $2 \leftrightarrow d$, $3 \leftrightarrow e$, $4 \leftrightarrow b$, $5 \leftrightarrow c$, $6 \leftrightarrow a$. Les nœuds dont le contour n'est *pas* en pointillés, associés aux arêtes épaisses, forment quant à eux un MCPS.

un ensemble fini de variables, D associe un domaine $D(x_i)$ à chaque variable $x_i \in X$, et C est un ensemble de contraintes. Chaque contrainte est définie sur un sous-ensemble de variables et donne la liste des valeurs qui peuvent être affectées à ces variables simultanément. Une solution est une affectation de toutes les variables qui satisfait toutes les contraintes.

[18] a introduit un modèle CSP pour le problème du MCIS se montrant plus performant que le *branch and bound* standard de [16]. Puisque le problème du MCIS est un problème d'optimisation, ce modèle utilise la généralisation des CSP connue sous le terme CSP *soft*. Dans un CSP souple, les solutions doivent satisfaire un ensemble de contraintes dites *dures* tout en optimisant une *fonction objectif*, qui correspond au coût de violation associé à une combinaison de contraintes *souples* (*soft*). Pour deux graphes G et G' donnés, ce modèle est :

— Variables : $X = \{x_u \mid u \in N_G\} \cup \{x_\perp, cost\}$
— Domaines : $D(x_\perp) = \{\perp\}$, $D(cost) = [0,|N_G|]$, et $\forall u \in N_G, D(x_u) = N_{G'} \cup \{\perp\}$
— Contraintes dures : $\forall \{u,v\} \subseteq N_G, (x_u = \perp) \vee (x_v = \perp) \vee ((u,v) \in E_G \Leftrightarrow (x_u,x_v) \in E_{G'})$
— Contraintes souples : *softAllDiff*$(X,cost)$

Chaque nœud u de G est associé à une variable x_u dont le domaine contient tous les nœuds de G'. Les domaines contiennent également une valeur supplémentaire $\perp$ qui est utilisée lorsque u n'est associée à aucun nœud de G'. Une variable additionnelle $x_\perp$ joue un rôle particulier : elle reçoit forcément l'unique valeur de son domaine, qui n'est autre que $\perp$. Grâce à cette variable et à la contrainte souple *softAllDiff*, chaque autre variable se verra affecter une valeur différente de $\perp$ à chaque fois que cela sera possible. La variable *cost* est contrainte d'être égale au nombre de variables qui devraient changer de valeur pour satisfaire une hypothétique contrainte *allDiff* dure. L'ensemble de contraintes dures garantit la préservation des arêtes dans le sous-graphe commun construit. Une solution de ce CSP est une affectation de *toutes* les variables de X qui satisfait chaque contrainte dure tout en minimisant la valeur de *cost*. Le MCIS associé à une solution de ce CSP souple est défini par l'ensemble de nœuds $u' \in N_{G'}$ tel qu'il existe une variable $x_u \in X$ qui est associée à u' dans cette solution. [18] évalue expérimentalement différentes techniques de propagation de contraintes. La combinaison « MAC+Bound » fourni généralement de très bons résultats : *« Maintaining Arc Consistency »* (MAC) [23] est utilisée pour propager les contraintes dures ; *« Bound »* vérifie qu'il est toujours possible d'affecter des valeurs distinctes à suffisamment de variables de X pour faire mieux que le plus faible coût (valeur de *cost*) rescencé parmi les sous-graphes communs déjà trouvés (c'est une version allégée de GAC(*softAllDiff*) [19] qui calcule le nombre maximal de variables qui pourront obtenir des valeurs distinctes).

2.3 Graphe de compatibilité

Une autre approche visant à résoudre le problème du MCIS est basée sur sa reformulation en instance du problème de la *clique maximum* dans un *graphe de compatibilité* [1, 8, 20].

Définition 9 *Une* clique *est un sous-graphe dont les nœuds sont tous liés deux à deux. Une clique est dite* maximale *si elle n'est pas strictement incluse dans une autre clique, et* maximum *s'il s'agit d'une des plus grandes cliques du graphe considéré, en prenant comme critère le nombre de nœuds qui la composent.*

Définition 10 *Un graphe de compatibilité pour le problème du MCIS entre deux graphes G et G' est un graphe G_C dont l'ensemble des nœuds correspond à $N_{G_C} = N_G \times N_{G'}$ et où deux nœuds sont reliés par une arête s'ils sont* compatibles. *Soient (u,u') et (v,v') deux nœuds de G_C (i.e. $\{u,v\} \subseteq N_G$ et $\{u',v'\} \subseteq N_{G'}$). (u,u') et (v,v') sont compatibles si $u \neq v$ et $u' \neq v'$, et si les arêtes sont préservées (i.e. $(u,v) \in E_G \Leftrightarrow (u',v') \in E_{G'}$).*

Comme chacun pourra le constater dans la figure 2, une clique dans G_C décrit un ensemble d'associations compatibles entre les nœuds de G et de G'. Ainsi, une telle clique correspond à un sous-graphe induit commun, et une clique maximum dans G_C est un MCIS de G et G'. Il suit logiquement que toute méthode capable de trouver une clique maximum dans un graphe peut être employée pour résoudre le problème du MCIS.

2.4 TR-décomposition

[10] a introduit une méthode de décomposition, nommée TR-décomposition, visant à résoudre une instance de CSP binaire en divisant les domaines des

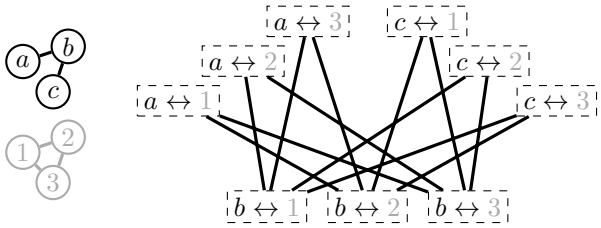


FIGURE 2 – Deux graphes et le graphe de compatibilité associé, qui souligne les combinaisons possibles d'affectations, par paires. Par exemple, puisque a et c ne sont reliés par aucune arête tandis que 1 et 3 le sont, les affectations $a \leftrightarrow 1$ et $c \leftrightarrow 3$ ne sont *pas* compatibles, car une arête serait amenée à disparaître, ce qui est formellement interdit par le problème du MCIS.

variables afin d'obtenir des sous-problèmes indépendants. Cette méthode s'appuie sur une réduction du CSP en une instance du problème de la recherche d'une clique de taille n (où n est le nombre de variables du CSP) dans la microstructure du CSP. La microstructure d'un CSP est un graphe similaire au graphe de compatibilité décrit en section 2.3. Chaque nœud de la microstructure correspond à une affectation, donc un couple $(variable, valeur)$. Deux nœuds sont reliés par une arête si et seulement si les affectations correspondantes sont compatibles, i.e. si chaque contrainte du problème est satisfaite lorsque ces deux affectations sont effectuées simultanément. De ce fait, une clique à n sommets dans la microstructure constitue une solution au CSP.

L'idée principale derrière la TR-décomposition est de trianguler la microstructure afin de générer plus aisément les sous-problèmes : un graphe quelconque peut comporter un nombre exponentiel de cliques maximales en fonction du nombre de nœuds. Un graphe triangulé, en revanche, ne peut avoir plus de $|N_G|$ cliques maximales, et la structure spécifique de ces graphes facilite également l'énumération de ces cliques [9]. [10] considère la classe des graphes triangulés afin de faire bénéficier la TR-décomposition de ces propriétés.

Définition 11 *Un graphe G est triangulé si et seulement si chacun de ses cycles de longueur supérieure ou égale à* 4 *possède au moins une* corde. *On appelle corde d'un cycle toute arête dont les extrémités sont deux nœuds* non consécutifs *du cycle.*

On appelle *triangulation* l'opération qui consiste à transformer un graphe donné en graphe triangulé *uniquement en ajoutant des arêtes.* La figure 3 montre une triangulation possible du graphe de compatibilité déjà rencontré dans la figure 2.

Puisque la triangulation ajoute des arêtes sans jamais en enlever, les cliques maximales du graphe

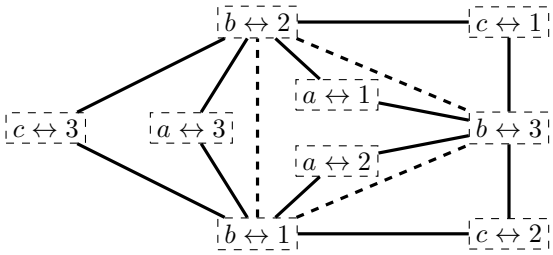


FIGURE 3 – Une version triangulée de graphe de compatibilité de la figure 2. Les arêtes qui ont dû être ajoutées (*« fill edges »*) sont indiquées en pointillés.

d'origine restent présentes au cours de la transformation : chacune est incluse dans une clique maximale du graphe triangulé. L'étape suivante de la TR-décomposition consiste à extraire ces cliques se trouvant au sein des nouvelles cliques maximales. Cette tâche est généralement plus aisée lorsque les cliques maximales du graphe triangulé sont petites, et donc plus proches des cliques d'origine.

En résumé, la méthode appelée TR-décomposition consiste à construire la microstructure du problème, à la trianguler, à extraire les cliques maximales de la microstructure triangulée, et à résoudre séparément les sous-problèmes induits par ces cliques, ce qui permet de retrouver les cliques de la microstructure d'origine.

3 Méthode de décomposition pour le problème du MCIS

Nous proposons ici de combiner la TR-décomposition avec le modèle de PPC défini dans [18] afin de résoudre le problème du MCIS en utilisant le graphe de compatibilité. Tandis que la TR-décomposition avait initialement été définie pour des instances de CSP, le modèle de PPC de [18] est un CSP *souple*. Une adaptation est donc nécessaire. Il se trouve que le problème du MCIS peut être résolu en trouvant une clique maximum dans le graphe de compatibilité. Nous inspirant de [10], nous suggérons d'utiliser la classe des graphes triangulés afin de tirer profit de leur nombre de cliques maximales limité.

Dans ce but, nous construisons le graphe de compatibilité de l'instance de problème du MCIS et triangulons ce graphe, ajoutant donc des arêtes. Ces arêtes, appelées *« fill edges »*, ajoutent des compatibilités erronées, car elles sont en contradiction avec la définition du graphe de compatibilité. De ce fait, une clique maximum dans ce nouveau graphe n'est pas forcément la meilleure solution. Ce n'est qu'un *sous-problème* – parmi d'autres – dans lequel il est possible que nous trouvions plusieurs solutions. Une clique maximum du

graphe de compatibilité reste une clique (mais pas forcément maximum ni même maximale) dans le graphe triangulé. De plus, cette clique apparaîtra forcément dans au moins une des cliques maximales de ce graphe triangulé. Pour ces raisons, chaque clique maximale du graphe triangulé définit un sous-problème (par le biais du sous-graphe qu'elle induit dans le graphe de compatibilité d'origine) dans lequel il est possible de trouver une clique maximum du graphe de compatibilité initial. De tels sous-problèmes peuvent être vus comme des instances habituelles du problème du MCIS pouvant être résolues en appliquant le modèle de PPC de [18], à la différence près que les domaines des variables sont réduits.

Plus précisément, pour chaque clique maximale K relevée dans la version triangulée du graphe de compatibilité associé à G et G', on définit un sous-problème ayant les mêmes variables et contraintes (décrites en section 2.2) que le problème initial. Les domaines, en revanche, sont restreints à

$$D(x_u) = \{v \in N_{G'} \mid (u \leftrightarrow v) \in K\} \cup \{\bot\}$$

Les domaines respectifs de $x_\bot$ et de *cost* restent quant à eux inchangés. Ainsi, par exemple, dans la figure 3, le sous-ensemble de nœuds $\{(b \leftrightarrow 1), (b \leftrightarrow 3), (c \leftrightarrow 2)\}$ est une clique maximale. Dans le sous-problème associé, $D(b) = \{1,3,\bot\}$, $D(c) = \{2,\bot\}$, et $D(a) = \{\bot\}$.

Chaque solution à un des sous-problèmes correspond à une clique maximale du graphe de compatibilité, et donc à un sous-graphe induit commun. Le sous-problème de l'exemple précédent possède deux solutions : associer b à 1 et c à 2, ou associer b à 3 et c à 2. Ces solutions correspondent à autant de cliques maximales du graphe de compatibilité : $\{(b \leftrightarrow 1), (c \leftrightarrow 2)\}$ et $\{(b \leftrightarrow 3), (c \leftrightarrow 2)\}$. L'arête $\{(b \leftrightarrow 1), (b \leftrightarrow 3)\}$ n'est pas utilisée pour construire ces cliques-solutions car il s'agit d'une *fill edge*, ajoutée lors du processus de triangulation et n'étant donc pas présente dans le graphe de départ.

Notez que ces sous-problèmes sont *complètement indépendants*, une caractéristique qui permet une implémentation parallèle extrêmement aisée du processus de résolution. Toute méthode capable de résoudre un CSP ou le problème de la clique maximum peut être employée pour résoudre ces sous-problèmes.

3.1 Bornes inférieures

Au cours de la résolution des sous-problèmes (que celle-ci se déroule en séquentiel ou en parallèle), il est possible d'ignorer tout sous-problème au sujet duquel on peut démontrer qu'il ne contient pas de solutions plus grandes que la meilleure trouvée jusque-là. En effet, certains sous-problèmes comportent des domaines contenant uniquement la valeur $\bot$. Une variable ayant un tel domaine n'aura d'autre choix que de se voir affecter cette valeur spéciale. Ainsi, si le nombre de variables ayant un domaine contenant autre chose que $\bot$ est plus petit que la taille du plus grand sous-graphe induit commun trouvé jusque-là, le sous-problème peut être ignoré. Par exemple, dans la figure 3, résoudre le sous-problème correspondant à la clique maximale $\{(b \leftrightarrow 1), (b \leftrightarrow 3), (c \leftrightarrow 2)\}$ fournit un MCIS potentiel à deux nœuds. Une fois cette solution trouvée, il est inutile d'examiner les sous-problèmes dans lesquels seules deux variables ou moins ont un domaine n'étant *pas* restreint à $\bot$: de tels sous-problèmes ne peuvent contenir de solutions comportant plus de deux nœuds et ne nous permettront donc pas de faire mieux que la solution précédemment trouvée.

Bien entendu, il est intéressant de trouver de grands sous-graphes induits communs tôt, car cela permet d'éliminer davantage de sous-problèmes. Pour ce faire, il est possible d'exécuter un algorithme heuristique afin d'obtenir rapidement une première solution, non optimale mais satisfaisante, avant de procéder à la décomposition. Dans nos expériences, nous utilisons *AntClique* [25] (disponible à l'adresse [24]) afin de rechercher des cliques maximales dans le graphe de compatibilité initial. Il est apparu que cette heuristique était capable de trouver une solution optimale pour plus de 90 % des instances résolues, malgré un temps d'exécution souvent négligeable par rapport au temps complet de résolution. Notez qu'*AntClique* ne prouve pas l'optimalité ; cette tâche est laissée à nos algorithmes de résolution de CSP.

Éliminer des sous-problèmes de cette manière était particulièrement important dans l'approche présentée par [3], car le CSP considéré était strict. Tout sous-problème avec au moins une variable au domaine vide pouvait être ignoré, car seules les solutions affectant une valeur à *chaque* variable étaient valides. Dans notre cas (qui est un problème d'optimisation), en revanche, les solutions valides peuvent se contenter de ne donner une valeur qu'à un sous-ensemble des variables. La seule manière d'éliminer un sous-problème en ayant la certitude de ne pas nuire à la complétude de l'approche est donc de prouver qu'il ne comporte pas de solution plus grande que la meilleure solution actuelle. Dans de telles conditions, il faut qu'un nombre significatif de valeurs aient un domaine vide. De plus, il se trouve que décomposer ce problème en se contentant de trianguler donne rarement lieu à de telles possibilités d'éliminations, dans les expériences que nous menons.

3.2 Complexité

Soient n et n' les nombres respectifs de nœuds des deux graphes dans lesquels nous recherchons un MCIS.

La première étape consiste à construire le graphe de compatibilité, chose faite en temps quadratique par rapport à son nombre de nœuds, i.e., en $\mathcal{O}((nn')^2)$. Nous lançons ensuite *AntClique* afin d'obtenir une borne inférieure de la taille du MCIS. Cet algorithme demande un temps en $\mathcal{O}((nn')^2)$, car le nombre de cycles d'*AntClique* est borné. La troisième partie de la résolution est la triangulation du graphe de compatibilité. Le but est ici d'ajouter le moins d'arêtes possible. Il se trouve qu'il s'agit d'un problème $\mathcal{NP}$-difficile. Nous sommes donc contraints d'utiliser une heuristique, en l'occurrence l'algorithme glouton *minFill* [12], qui nous permet d'effectuer cette triangulation en $\mathcal{O}((nn')^3)$. La quatrième étape est l'extraction des cliques maximales du graphe triangulé. Il y en a au plus nn', et chacune d'entre elles est obtenue en temps linéaire. Enfin, la dernière étape est de résoudre les sous-problèmes représentés par ces cliques maximales. L'initialisation du problème en fonction des sommets composant la clique demande un temps linéaire par rapport au nombre d'arêtes de cette clique. Chaque sous-problème a $\mathcal{O}(n)$ variables. Cependant, la taille des domaines dépend, elle, de la structure du graphe de compatibilité triangulé. Soit d la taille du plus grand domaine d'un sous-problème (avec $d \leq n' + 1$). Chaque sous-problème est résolu avec MAC+Bound, en $\mathcal{O}(d^3.n^2.d^n)$. Il en suit que la complexité temporelle globale de la TR-décomposition est en $\mathcal{O}((nn')^3 + nn'(d^3.n^2.d^n))$.

En comparaison, la complexité du problème du MCIS sans décomposition est en $\mathcal{O}(d'^3.n^2.d'^n)$ où d' est la taille du plus grand domaine dans le problème initial. Il faut noter que $d \leq d'$ et que selon la qualité de la décomposition la valeur de d peut être beaucoup plus petite que celle d'.

4 Premières expériences

Les expériences suivantes ont pour but de comparer l'approche de PPC pure de [18] (*MAC+Bound*) avec la TR-décomposition. Ces deux méthodes utilisent la même heuristique *AntClique* pour obtenir une solution initiale, qui sert de première borne inférieure.

Nos algorithmes ont été développés en C et compilés avec GCC, en utilisant le niveau d'optimisation `-O3`. Les programmes ont été exécutés avec une limite de temps de trois heures sur des processeurs Intel® Xeon® CPU E5-2670 0 à 2,60 GHz, avec 20 480 Ko de mémoire cache et 4 Go de RAM.

Les instances utilisées proviennent du *benchmark* présenté par [4] et ont été prises dans leur version étiquetée et orientée. Le nombre d'étiquettes différentes, pour une instance donnée, a été fixé à 15 % du nombre de nœuds des deux graphes de l'instance. Par exemple, quand les graphes ont 60 nœuds, il y a $60 \times 15 \div 100 = 9$ types d'étiquettes différents. ce paramètre permet de régler la difficulté des instances. Ces étiquettes sont prises en compte comme expliqué dans [18], en s'assurant que des nœuds associés pour construire un sous-graphe commun possèdent la même étiquette. De manière similaire, les arêtes situées entre des paires de nœuds correspondants doivent avoir la même étiquette. Chaque domaine est ainsi réduit aux valeurs associées à des nœuds ayant la même étiquette que le nœud associé à la variable.
Nous considérons trois types d'instances :

bvg *Bounded Valence Graphs*, dans lesquels les degrés des sommets ne peuvent dépasser une valeur fixe, propre au graphe.

rand *Random graphs*, dans lesquels les arêtes sont disposées de manière aléatoire (voir [4] pour plus de détails).

m2D, m3D and m4D *Meshes* (maillages) à deux, trois ou quatre dimensions respectivement.

Ces graphes ont un nombre de nœuds allant de 10 à 100 inclus. Nous considérons 3 268 instances prises au hasard parmi les 81 700 que présente le *benchmark*, sans favoriser l'un des types d'instances. Leurs proportions sont ainsi relativement préservées.

Puisque notre méthode de décomposition est avant tout conçue pour les instances particulièrement difficiles, nous nous concentrons sur les (nombreuses) instances que la méthode standard (*MAC+Bound* [18]) ne parvient pas à résoudre en moins de 100 secondes. Le nombre d'instances passe ainsi de 3 268 à seulement 1 177.

Le tableau 1 donne les spécificités des instances se trouvant dans les séries que nous avons utilisées, et ce avant et après le filtrage retirant les instances jugées trop faciles selon le critère susmentionné. On remarque que les graphes aléatoires (*rand*) sont moins présents après filtrage, ce qui signifie qu'ils ont tendance à être plus faciles. A contrario, les maillages représentent une part bien plus importante de notre panel d'instances après ce filtrage. Cela n'a rien d'une surprise, car les maillages sont connus pour rendre difficile la recherche d'un plus grand sous-graphe commun, et ceux-ci ont été créés par [4] principalement pour cette raison. Cette difficulté provient de la structure répétitive de ces graphes, qui fait apparaître de nombreux grands sous-graphes communs qui ne sont cependant que rarement optimaux.

Nous allons dans un premier temps évaluer la qualité

Tableau 1 – La proportion représentée par chaque type de graphe dans l'ensemble d'instances de départ, puis après exclusion des instances résolues en moins de 100 secondes sans décomposition.

	Toutes les instances	Instances difficiles
Total	3 268	1 177
bvg	1 232 (37,7 %)	394 (33,5 %)
rand	744 (22,8 %)	135 (11,5 %)
m2D	700 (21,4 %)	335 (28,5 %)
m3D	384 (11,8 %)	201 (17,1 %)
m4D	208 (6,4 %)	112 (9,5 %)

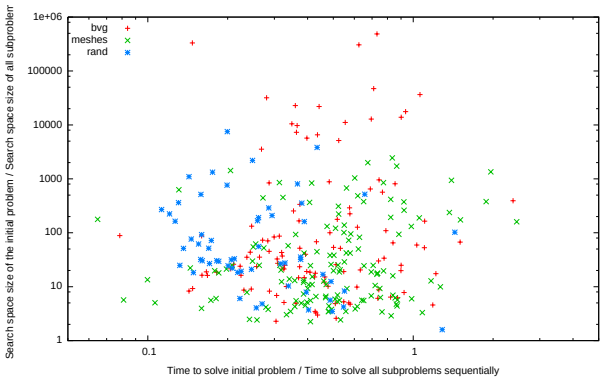


Figure 4 – La réduction de la taille de l'espace de recherche mise en relation avec la réduction du temps de calcul. Chaque point (x,y) correspond à une instance i. Soit S_i l'ensemble des sous-problèmes générés à partir de i. x donne le rapport entre le temps de calcul nécessaire à la résolution de i et celui nécessaire à la résolution de tous les sous-problèmes de S_i (sur un unique processeur, et en incluant le temps de décomposition). y donne le rapport entre l'espace de recherche de i et la somme des espaces de recherche des sous-problèmes de S_i.

de la décomposition par rapport à la taille des espaces de recherche avec et sans décomposition. La taille de l'espace de recherche d'un (sous-)problème est égale à la taille du produit cartésien de ses domaines, en comptant $\bot$. Dans le cadre de la TR-décomposition, la somme des tailles des espaces de recherche des différents sous-problèmes est une bonne mesure de la qualité de la décomposition dans le cas extrême où tous les sous-problèmes sont résolus consécutivement. Notre premier constat est que la TR-décomposition réduit l'espace de recherche. Nous étudions donc tout d'abord l'impact de cette réduction sur le processus de résolution. La figure 4 nous montre que cette réduction est d'un facteur généralement situé entre 2 et 10^6. Cependant, cette réduction ne se traduit pas systématiquement par une diminution du temps de calcul nécessaire à la résolution. Dans la figure 4, nous comparons le temps nécessaire à la résolution du problème initial et le temps nécessaire à la décomposition du problème et à la résolution de tous les sous-problèmes sur *un seul* processeur. Nous notons que seules 8 (resp. 10 et 2) instances de type *bvg* (resp. *meshes* et *rand*) sont résolues plus rapidement, dans ces conditions de processeur unique, avec la décomposition. Il s'agit d'un point négatif de notre approche, et nous comptons étudier cet inconvénient avec davantage d'attention.

Cependant, du fait que les sous-problèmes sont indépendants, nous pouvons les résoudre en parallèle plutôt que sur un unique processeur. On remarque que pour de nombreuses instances, il n'y a qu'un nombre très faible de gros sous-problèmes, entourés d'une multitude de sous-problèmes de taille réduite. De ce fait, il n'est pas extrêmement utile d'avoir autant de processeurs à notre disposition que nous avons généré de sous-problèmes. Si le nombre de processeurs est limité à m, nous pouvons, une fois la décomposition terminée, disposer les sous-problèmes dans une file (une méthode similaire à ce qui a été fait dans [21, 22]). Un sous-problème est initialement donné à chaque processeur. Par la suite, dès lors qu'un processeur a terminé la résolution du sous-problème qui lui avait été attribué, le sous-problème suivant dans la file lui est affecté. Pour améliorer la répartition de la charge de travail, nous tentons de placer les sous-problèmes semblant les plus complexes en premier dans la file, en utilisant la taille de l'espace de recherche comme critère d'estimation de la difficulté. Ainsi, la figure 5 montre que seuls 9 processeurs suffisent à obtenir des performances comparables avec celles obtenues avec un nombre illimité de processeurs.

Avec 9 processeurs, notre décomposition est généralement plus rapide que la méthode sans décomposition, puisqu'une fois la décomposition effectuée, le temps nécessaire au reste du processus de résolution tend à être égal ou proche du temps nécessaire à la résolution du plus difficile des sous-problèmes. De plus, sur ces instances, le temps de décomposition est généralement négligeable.

Étant donné que nous tentons ici d'employer un nombre réduit de processeurs, il est préférable de conserver le nombre de sous-problèmes le plus bas possible, et ce sans annihiler les bénéfices de la décomposition. Le nombre moyen de sous-problèmes est de 33,44 sur ces instances, avec un maximum de 142. Notez cependant que seules trois instances dépassaient le nombre de 100 sous-problèmes. Ainsi, le nombre de sous-problèmes à résoudre peut rendre notre nombre de processeurs insuffisant pour certaines instances.

Le coefficient d'accélération obtenu en utilisant m

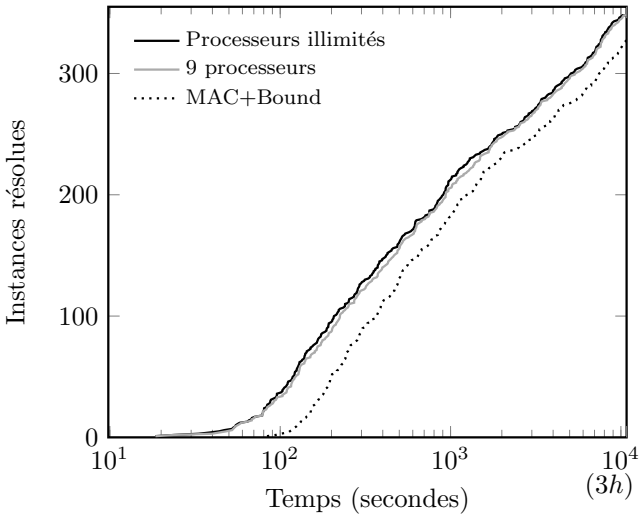


FIGURE 5 – Une comparaison du nombre d'instances résolues pour différentes limites de temps (il ne s'agit pas du cumul des temps) pour la méthode standard et pour la TR-décomposition, avec un nombre illimité hypothétique de processeurs. Une troisième courbe montre que 9 processeurs seraient suffisants pour résoudre autant d'instances dans la limite du temps imparti.

processeurs se trouve être inférieur à m. cette différence s'explique par la présence de nombreuses redondances entre sous-problèmes : différents processeurs font des vérifications similaires, perdant ainsi du temps.

Résoudre les sous-problèmes en parallèle réduit l'impact de ces redondances, mais ce point souligne le fait qu'utiliser la TR-décomposition avec un unique processeur (ou un nombre très faible par rapport au nombre de sous-problèmes générés) à notre disposition donnerait des résultats médiocres lorsque les sous-problèmes ont de grandes intersections, car un même processeur pourrait résoudre les mêmes parties du problème à plusieurs reprises. Pour cette raison, il peut être utile d'ajouter une phase de prétraitement visant à réduire la taille des intersections entre sous-problèmes.

5 Réduction du nombre de sous-problèmes par fusion

Bon nombre de couples de sous-problèmes présentent de grandes intersections, ce qui signifie que nous risquons de résoudre plusieurs fois des parties similaires du problème : le même sous-graphe commun serait construit par deux fils d'exécution sans que nous n'en tirions aucun bénéfice. Afin de limiter ces redondances tout en réduisant le nombre de sous-problèmes, il est possible de *fusionner* certains de ces sous-problèmes. Plus formellement, deux sous-problèmes peuvent être remplacés par un nouveau sous-problème en calculant l'union des valeurs autorisées par les deux sous-problèmes d'origine. Notez que les solutions se trouvant dans les sous-problèmes initiaux sont toujours présentes dans la version fusionnée, car les domaines des variables ne peuvent que grandir durant cette opération.

Dans le but d'empêcher les sous-problèmes de grossir jusqu'à revenir à une taille comparable à celle du problème initial non décomposé, ce qui rendrait la TR-décomposition inutile, il est possible de fixer une limite, en prenant par exemple comme critère la taille de l'espace de recherche du sous-problème formé par la fusion. Dans de telles conditions, la fusion de deux sous-problèmes est interdite si cela créerait un sous-problème dont la taille de l'espace de recherche dépasse la limite choisie. Cette limite peut être déterminée en fonction de la plus grande taille d'espace de recherche trouvée parmi les sous-problèmes. L'idée derrière ce choix est d'empêcher l'apparition de sous-problèmes qui seraient à même de ralentir le processus global de résolution, puisque c'est souvent le sous-problème le plus difficile qui se montre déterminant pour le temps de résolution en parallèle. Il est possible de combiner cette limite avec d'autres. Nous utilisons par exemple une comparaison de la somme des tailles d'espaces de recherche des deux problèmes considérés et de celle du sous-problème qui résulterait d'une fusion de ces deux sous-problèmes. Si la somme calculée est bien plus grande que cette taille qui pourrait être obtenue, cela signifie que l'intersection des espaces de recherche des deux sous-problèmes considérés a une taille non négligeable et qu'il serait intéressant de les fusionner.

Pour résumer ce principe, nous introduisons ici la notion de *gain*, qui exprime l'évolution de la taille totale de l'espace de recherche au cours d'une fusion hypothétique :

Définition 12 *Soient S_1 et S_2 deux sous-problèmes distincts. Le* gain *offert par la fusion de S_1 et de S_2 est donné par le rapport suivant : $gain(S_1, S_2) = \frac{size(S_1)+size(S_2)}{size(S_1 \cup S_2)}$, où $size(S_i)$ correspond à la taille de l'espace de recherche de S_i, et $S_1 \cup S_2$ représente le sous-problème qui serait créé si S_1 et S_2 venaient à être fusionnés.*

Il s'ensuit logiquement que deux sous-problèmes offrant un gain strictement supérieur à 1, une fois fusionnés, permettent à la complexité théorique d'être plus basse qu'en résolvant ces sous-problèmes séparément. À l'inverse, un gain strictement inférieur à 1 signifie que fusionner ces sous-problèmes augmenterait

la complexité théorique.

Bien entendu, fusionner des sous-problèmes lorsque leur nombre a déjà été ramené au nombre de processeurs disponibles ou à un nombre encore plus faible ne présente a priori aucun avantage. Des fusions supplémentaires risqueraient de créer des sous-problèmes extrêmement difficiles, car les limites utilisées, basées sur les tailles d'espaces de recherche, ne sont pas infaillibles et sont avant tout des moyens d'estimations de la difficulté. Nos expériences nous ont montré que ce risque vaut certes la peine d'être pris lorsque le nombre de sous-problème est grand mais qu'il valait mieux stopper le processus de fusion dès que ce nombre devenait convenablement faible.

6 Expériences avec fusions de sous-problèmes

Ces nouvelles expériences emploient la fusion de sous-problèmes afin d'effectuer un prétraitement. Le but principal de ces fusions est de réduire l'impact des intersections entre sous-problèmes : fusionner deux sous-problèmes permet d'éviter que le programme ne fasse de mêmes vérifications plusieurs fois. Cependant, fusionner de manière irréfléchie peut ralentir la résolution dans son ensemble en créant des sous-problèmes exceptionnellement longs à résoudre.

Au cours de ces expériences, les restrictions suivantes ont été imposées lors du prétraitement :

— Le gain minimal qu'une paire de sous-problèmes doit offrir pour prétendre à la fusion est de 1, excepté dans les cas où une autre valeur est explicitement indiquée.

— Toute fusion qui créerait un sous-problème ayant un espace de recherche plus grand que le plus grand observable parmi les sous-problèmes courants est interdite, comme expliqué en section 5.

— Nous n'effectuons qu'une fusion à la fois, en choisissant à chaque cycle la paire de sous-problèmes offrant le meilleur gain, tout en ignorant celles qui ne respectent pas la condition précédemment exposée.

Les résultats obtenus sont visibles dans les figures 6 et 7. Dans la limite de temps de 3 heures, plusieurs instances sont uniquement résolues par les méthodes usant de la TR-décomposition.

En ce qui concerne le nombre de sous-problèmes, nous observons une moyenne de 33,44 (resp. 31,69, 20,63 et 8,22) sur les 1 177 instances examinées si aucune fusion n'est effectuée (resp. si les sous-problèmes sont fusionnés avec un gain minimal de 1, 0,5 et 0). Cette moyenne n'a pas pu descendre en dessous de 8, même avec un gain minimal de 0 (auquel cas la seule limite fixée est celle relative à la taille de l'espace de recherche le plus grand parmi les sous-problèmes), car huit processeurs étaient utilisés et les fusions sont nécessairement stoppées dès lors qu'il ne reste pas plus d'un sous-problème par processeur. Il est également intéressant de noter que le nombre de sous-problèmes générés durant ces expériences correspondait généralement à moins de un pourcent du maximum théorique (qui est égal au nombre de nœuds dans le graphe de compatibilité).

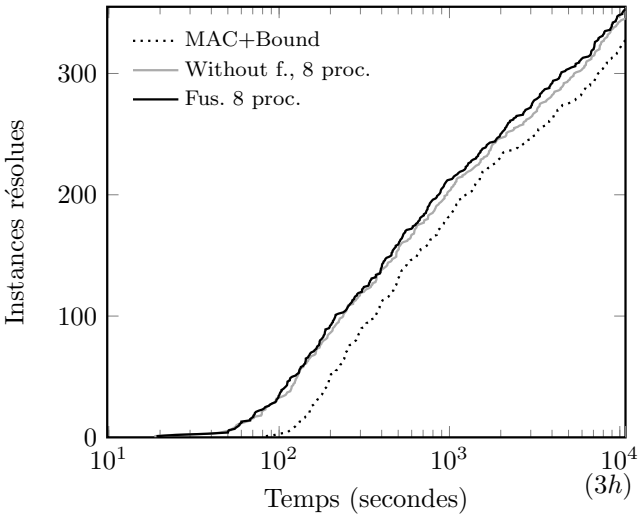


FIGURE 6 – Ce graphique montre notre *baseline* (un unique CSP résolu avec MAC et une borne inférieure) ainsi que deux autres courbes pour la TR-décomposition sur 8 processeurs : l'une avec fusions (avec un gain minimal de 1) et l'autre sans fusion.

Concernant le temps nécessaire pour réaliser les fusions, il n'a pas dépassé le tiers de seconde pour un gain minimal de 1 ou les deux tiers de seconde pour 0.

Le tableau 2 montre l'impact du choix de gain minimal de fusion sur les performances. Des fusions excessives ont un effet négatif sur certaines instances, tandis que ne pas fusionner du tout fait également baisser les performances.

La figure 6 montre qu'utiliser un gain minimal de fusion de 1 permet effectivement à la TR-décomposition d'être plus efficace. La tableau 2 résume ces résultats de manière plus lisible.

On constate que le choix d'un gain minimal de 1 offre de meilleurs résultats que les autres possibilités ici passées en revue.

Puisque la fusion de sous-problèmes est avant tout conçue pour diminuer les redondances entre sous-problèmes, cette méthode peut rapprocher de n le coefficient d'accélération offert par l'usage de n processeurs. Notez cependant que, malgré ces redondances, 25 instances ont été résolues plus rapidement avec dé-

Tableau 2 – Le nombre d'instances résolues après différents laps de temps, pour différentes politiques de fusion : g0 n'a pour seule limite que la règle interdisant de créer des sous-problèmes plus grands que le plus grand courant ; g0,5 ignore les paires de sous-problèmes offrant un gain inférieur à 0,5 ; g1 est notre méthode déjà présentée dans les figures précédentes ; « S. f. » n'effectue aucune fusion. « M+B » correspond aux résultats obtenus avec MAC+Bound.

T (s)	M+B	F. g0	F. g0,5	F. g1	S. f.
0	0	0	0	0	0
100	2	21	29	34	32
200	50	81	84	92	86
400	111	132	135	140	138
800	163	187	190	196	185
1 600	220	233	233	238	231
3 200	249	269	270	278	268
6 400	289	306	309	313	310
10 800	327	347	351	353	345

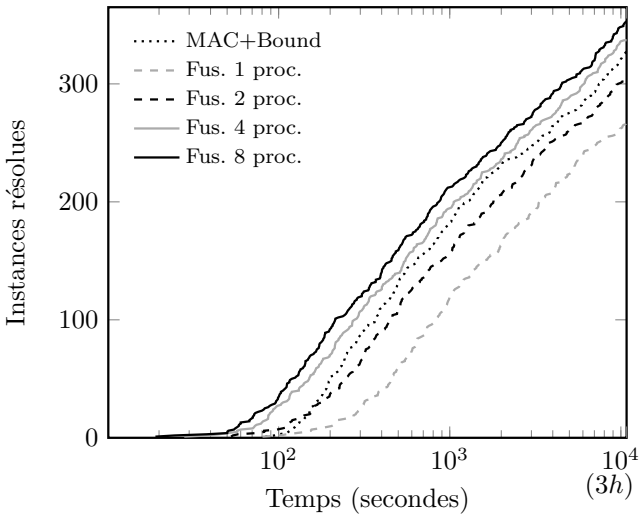


Figure 7 – Ce graphique montre l'évolution des performances pour différents nombres de processeurs. Les fusions sont ici toujours activées (gain minimal de 1). La *baseline*, présente dans les figures précédentes, est également affichée.

composition même en utilisant un unique processeur, parfois plus de deux fois plus rapidement.

7 Conclusion

Nous présentons dans cet article une adaptation de la TR-décomposition, préalablement définie par [10], au problème du plus grand sous-graphe commun induit. Cette méthode s'appuie sur une triangulation du graphe de compatibilité du problème et permet de décomposer ce problème en sous-problèmes indépendants, pouvant être résolus en parallèle. Nous présentons une technique de fusion visant à réduire les redondances inhérentes à cette approche. Les expériences montrent que la TR-décomposition est une méthode adaptée au problème du MCIS sur les instances difficiles. Sur ces instances, le temps nécessaire à la décomposition est généralement rentabilisé grâce à un raccourcissement significatif de la phase de résolution proprement dite. De plus, fusionner des sous-problèmes de manière avisée ouvre de nouvelles perspectives et améliorent ces résultats.

À l'avenir, plusieurs points feront l'objet d'une attention plus particulière.

D'autres méthodes de décomposition présentent un certain attrait, comme effectuer plusieurs triangulations pour équilibrer les sous-problèmes. Là encore, il nous sera nécessaire de trouver un bon compromis entre le temps que nécessite la décomposition appliquée et sa qualité.

Il sera également possible d'adapter la TR-décomposition à l'approche consistant à recherche une clique maximale dans le graphe de compatibilité (intégral ou, avec des sous-problèmes, divisé) plutôt que de résoudre des CSP.

Il serait intéressant d'employer des instances plus réalistes, provenant de cas scientifiques réels ou tout du moins présentant des structures plus caractéristiques.

Références

[1] Egon Balas and Chang Sung Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4) :1054–1068, 1986.

[2] H. G. Barrow and R. M. Burstall. Subgraph isomorphism, matching relational structures and maximal cliques. *Information Processing Letters*, 4(4) :83–84, 1976.

[3] Assef Chmeiss, Philippe Jégou, and Lamia Keddar. On a generalization of triangulated graphs for domains decomposition of csps. In *IJCAI*, pages 203–208. Citeseer, 2003.

[4] Donatello Conte, Pasquale Foggia, and Mario Vento. Challenging complexity of maximum common subgraph detection algorithms : A performance analysis of three algorithms on a wide database of graphs. *J. Graph Algorithms Appl.*, 11(1) :99–143, 2007.

[5] Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1) :5–41, 2001.

[6] Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. *Artificial intelligence*, 171(2) :73–106, 2007.

[7] Matjaz Depolli, Janez Konc, Kati Rozman, Roman Trobec, and Dusanka Janezic. Exact parallel maximum clique algorithm for general and protein graphs. *Journal of chemical information and modeling*, 53(9) :2217–2228, 2013.

[8] Paul J Durand, Rohit Pasari, Johnnie W Baker, and Chun-che Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2(17) :1–16, 1999.

[9] Delbert R Fulkerson, Oliver A Gross, et al. Incidence matrices and interval graphs. *Pacific J. Math*, 15(3) :835–855, 1965.

[10] Philippe Jégou. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *AAAI*, volume 93, pages 731–736, 1993.

[11] Philippe Jégou and Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146(1) :43–75, 2003.

[12] Uffe Kjaerulff. Triangulation of graphs - algorithms giving small total state space. Technical report, Judex R.R. Aalborg., Denmark, 1990.

[13] I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250(1) :1–30, 2001.

[14] Ciaran McCreesh and Patrick Prosser. Multi-threading a state-of-the-art maximum clique algorithm. *Algorithms*, 6(4) :618–635, 2013.

[15] Ciaran McCreesh and Patrick Prosser. The shape of the search tree for the maximum clique problem, and the implications for parallel branch and bound. *arXiv preprint arXiv :1401.5921*, 2014.

[16] James J McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software : Practice and Experience*, 12(1) :23–34, 1982.

[17] Maël Minot and Samba Ndojh Ndiaye. Searching for a maximum common induced subgraph by decomposing the compatibility graph. In *Bridging the Gap Between Theory and Practice in Constraint Solvers, CP2014-Workshop*, pages 1–17, September 2014.

[18] Samba Ndojh Ndiaye and Christine Solnon. Cp models for maximum common subgraph problems. In *Principles and Practice of Constraint Programming–CP 2011*, pages 637–644. Springer, 2011.

[19] Thierry Petit, Jean-Charles Régin, and Christian Bessière. Specific filtering algorithms for over-constrained problems. In *Principles and Practice of Constraint Programming – CP 2001*, pages 451–463. Springer, 2001.

[20] J W Raymond, E J Gardiner, and P Willett. Rascal : calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45(6) :631–644, 2002.

[21] Jean-Charles Régin, Mohamed Rezgui, and Arnaud Malapert. Embarrassingly parallel search. In *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, pages 596–610, 2013.

[22] Jean-Charles Régin, Mohamed Rezgui, and Arnaud Malapert. Improvement of the embarrassingly parallel search for data centers. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 622–635, 2014.

[23] Daniel Sabin and Eugene C Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Principles and Practice of Constraint Programming*, pages 10–20. Springer, 1994.

[24] Christine Solnon. Solving maximum clique problems with ant colony optimization liris.cnrs.fr/csolnon/antclique.html, 2006.

[25] Christine Solnon and Serge Fenet. A study of ACO capabilities for solving the maximum clique problem. *J. Heuristics*, 12(3) :155–180, 2006.

[26] Philippe Vismara. Programmation par contraintes pour les problemes de plus grand sous-graphe commun. In *JFPC'11 : Journées Francophones de Programmation par Contraintes*, pages 327–335, 2011.

Un cadre générique pour l'intégration de BTD dans une bibliothèque de programmation par contraintes

Samba Ndojh NDIAYE[1,2] **Christine SOLNON**[1,3]

[1] Université de Lyon, LIRIS, UMR 5205 CNRS
[2] Université Lyon 1, F-69622 France
[3] INSA-Lyon, F-69621, France

{samba-ndojh.ndiaye,christine.solnon}@liris.cnrs.fr

Résumé

La méthode BTD permet de résoudre efficacement des instances CSP en tirant profit de la structure des problèmes à travers une décomposition arborescente du réseau de contraintes. Cette décomposition permet d'identifier des sous-problèmes indépendants dont la résolution peut être effectuée par toute méthode de résolution de CSP classique. Dans cet article, nous introduisons un cadre générique pour intégrer BTD dans une bibliothèque de programmation par contraintes, ce qui nous permet de bénéficier des différents algorithmes de propagation et de recherche implémentés dans la bibliothèque. Ce cadre a été implémenté à l'aide de la bibliothèque Gecode, et nous considérons deux variantes : la première (BTD-G) utilise la recherche Gecode par défaut pour résoudre chaque sous-problème, tandis que la seconde (BTD-GR) utilise une recherche avec *restarts*. Cela nous permet d'évaluer l'intéret d'utiliser une recherche avec *restarts* lors de la résolution des sous-problèmes. Ces deux variantes sont comparées expérimentalement sur les instances de la compétition CSP'2009. Nous constatons que BTD-GR surpasse BTD-G pour beaucoup d'instances, et que BTD-GR surpasse Gecode (dans sa version avec *restarts*) pour un grand nombre d'instances structurées.

Abstract

BTD exploits tree decompositions of constraint networks to speed up their solution process. The decomposition is used to identify independent subproblems which may be solved by any constraint solver provided that it is complete. Since now, BTD has been implemented on top of *ad hoc* solvers which are often restricted to constraints defined in extension. In this paper, we introduce a framework for integrating BTD in a Constraint Programming library, thus allowing us to benefit from various kinds of constraints, propagators, and heuristics to solve subproblems identified by BTD. This framework has been implemented on top of Gecode, and we consider two variants of it : The first one (BTD-G) uses the default Gecode search without restarts, whereas the second one (BTD-GR) uses the restart-based Gecode search. This allows us to evaluate the interest of using restarts when assigning values to the variables of a same cluster. These two variants are experimentally evaluated on the CSP'2009 competition benchmark, showing that BTD-GR outperforms BTD-G on many instances, and that BTD-GR outperforms Gecode (with a restart policy) on many structured instances.

1 Introduction

Le formalisme CSP (problème de satisfaction de contraintes) permet de modéliser très simplement un problème en identifiant un ensemble de variables à affecter avec un ensemble de valeurs en essayant de respecter des contraintes. Les méthodes de résolution structurelles offrent les meilleures garanties théoriques pour la recherche d'une solution d'une instance CSP [6]. Cependant, leur efficacité pratique n'est pas toujours évidente à démontrer. La méthode BTD [11], tout en garantissant parmi les meilleures bornes de complexité théorique [10], a donné des résultats très prometteurs sur des instances binaires souvent générées de manière aléatoire [11, 9, 10]. Des évaluations ont également été menées avec succès sur des instances structurées de la compétition CSP-2008

[2, 13, 12, 15]. Malheureusement, ces dernières expérimentations posent des restrictions sur l'arité des contraintes, sur la définition des relations en extension ou en intention, voire sur la gestion des contraintes globales. Par ailleurs, le fait que BTD puisse reposer sur toute méthode capable de résoudre les sous-problèmes identifiés par la décomposition arborescente a conduit à plusieurs combinaisons ([11, 3, 2, 13]) avec des méthodes telles que FC [7], MAC [22], CBJ [20], Decision Repair [14, 19], mais aussi avec plusieurs heuristiques de choix de variables telles que dom/ddeg [1] et dom/wdeg [4]. [13] a proposé une intégration des politiques de restarts [5] dans BTD. Malgré tous ces travaux, il reste un grand nombre de possibilités qui n'ont pas encore été explorées et beaucoup de types d'instances sur lesquelles cette méthode n'a pas été évaluée. Il semble difficile d'accomplir cette tâche en implémentant directement tous les outils nécessaires du fait du nombre impressionnant de techniques proposées chaque année par la communauté programmation par contraintes. Néanmoins, il faut noter que cette communauté dispose de plusieurs bibliothèques regroupant une grande partie des techniques les plus performantes de résolution de CSP. Nous proposons de nous appuyer sur toutes ces ressources offertes par ces bibliothèques afin de faciliter l'intégration dans BTD des meilleures techniques de résolution de CSP et l'évaluation de ces combinaisons sur un ensemble diversifié d'instances contenues dans les benchmarks disponibles. Pour cela, nous définissons un cadre générique d'intégration de BTD dans une bibliothèque de programmation par contraintes (CP). Ce cadre suppose uniquement l'existence dans la bibliothèque d'une méthode capable d'énumérer l'ensemble des solutions d'une instance CSP. Nous montrons également comment étendre ce cadre à des approches de résolution utilisant des politiques de *restarts*. Ces deux variantes ont été implémentées dans la bibliothèque Gecode [23] et des expérimentations ont été menées sur l'ensemble des instances CSP du benchmark de la compétition CSP'2009. Les résultats démontrent l'efficacité de BTD combinée avec une politique de restarts limitée aux clusters de la décomposition arborescente.

La section 2 fait quelques rappels sur les notions de base liées à la méthode BTD. Dans la section 3, nous présentons le cadre générique, puis nous discutons de l'intégration de politiques de restarts dans la section 4. La section 5 décrit les résultats expérimentaux.

2 Préliminaires

Un *problème de satisfaction de contraintes* (X, D, C) est défini par un ensemble de variables X, une fonction domaine D qui associe à chaque variable $x_i \in X$ l'ensemble $D(x_i)$ des valeurs pouvant être affectées à x_i, et un ensemble de contraintes C à satisfaire. Une contrainte $c_j \in C$ est définie sur un sous-ensemble des variables $\mathit{portée}(c_j) \subseteq X$ et restreint les valeurs que ces variables peuvent prendre simultanément. Etant données deux fonctions domaine D et D', nous notons $D' \subseteq D$ si $D'(x_i) \subseteq D(x_i)$ pour chaque variable $x_i \in X$. Nous disons qu'une fonction domaine D affecte une variable x_i si $|D(x_i)| = 1$. Une solution est une fonction domaine qui affecte toutes les variables et satisfait toutes les contraintes.

La structure d'un CSP (X, D, C) peut être représentée par l'hypergraphe $H_{(X,D,C)} = (X, \mathcal{E})$, appelé *hypergraphe de contraintes*, tel que $\mathcal{E} = \{\mathit{portée}(c) | c \in C\}$. Chaque sommet de cet hypergraphe est associé à une variable du CSP, et chaque hyperarête à une contrainte. La 2-section de $H_{(X,D,C)}$ est le graphe $G_{(X,D,C)} = (X, E)$ où $E = \{\{x_i, x_j\} \subseteq X : \exists c_k \in C, \{x_i, x_j\} \subseteq \mathit{portée}(c_k)\}$. Ainsi, toute contrainte $c_k \in C$ induit une clique dans la 2-section qui contient les sommets associés à $\mathit{portée}(c_k)$. Une décomposition arborescente d'un CSP est définie sur la 2-section de son hypergraphe de contraintes.

Définition 1 *[21] Une décomposition arborescente d'un graphe $G = (X, E)$ est un couple (K, T) où $T = (I, F)$ est un arbre, $K : I \to \mathcal{P}(X)$ est une fonction qui associe un sous-ensemble de variables $K_i \subseteq X$ (appelé* cluster*) à chaque sommet, et les conditions suivantes sont vérifiées :*
(i) $\cup_{i \in I} K_i = X$;
(ii) pour chaque arête $\{x_j, x_k\} \in E$, il existe un sommet $i \in I$ tel que $\{x_j, x_k\} \subseteq K_i$;
(iii) pour tout $i, j, k \in I$, si k est sur le chemin entre i et j dans T, alors $K_i \cap K_j \subseteq K_k$.
La largeur w d'une décomposition arborescente (K, T) est égale à la taille maximum des clusters moins 1, i.e.*, $w = \max_{i \in I} |K_i| - 1$. La largeur d'arbre (ou* treewidth*) w^* de G est la largeur minimum sur l'ensemble de ses décompositions arborescentes.*

Etant donnée une décomposition arborescente (K, T) d'un CSP (X, D, C) et une racine $r \in I$, BTD identifie des sous-problèmes indépendants qui sont résolus séparément. Plus précisément, chaque sous-problème ne contient qu'un sous-ensemble de l'ensemble initial de variables. BTD affecte en premier les variables du cluster racine K_r. S'il n'existe pas d'affectation cohérente de ces variables, alors le problème est incohérent. Sinon, si r a k fils $i_1, \ldots, i_k$ dans l'arbre T, BTD résoud récursivement k sous-problèmes indépendants : chaque sous-problème est associé à un fils i_j de r et contient l'ensemble des variables apparaissant dans les clusters associés aux sommets du sous-arbre de T enraciné en i_j. Ces k sous-problèmes sont indé-

Algorithme 1 – BTD$((X, D, C), (K, T), r)$

Données : une instance CSP (X, D, C), une décomposition arborescente (K, T), et un sommet racine r de T
Résultats : Retourne *succès* s'il existe une fonction domaine $D' \subseteq D$ qui affecte de façon cohérente toutes les variables des clusters associés aux sommets de T ; retourne *échec* sinon.

```
1  model ← buildModel((X, D, C), K_r)
2  tant que model.next() ≠ null faire
3      Soit newD la fonction domaine retournée par next() (telle que newD affecte de façon cohérente toutes les variables de K_r)
4      Pour chaque fils i de r, soit A_i le tuple de valeurs affectées aux variables de K_r ∩ K_i dans newD
5      tant que il n'existe pas de fils j de r tel que A_j est un nogood
6      et il existe un fils i de r tel que A_i n'est pas un good faire
7          Soit T_i le sous-arbre de T enraciné en i
8          si BTD((X, newD, C), (K, T_i), i) = succès alors
9              enregistrer A_i comme good
10         sinon
11             enregistrer A_i comme nogood
12     si pour chaque fils i de r, A_i est un good alors
13         retourne succès
14 retourne échec
```

pendants car, dès lors que les variables du cluster racine K_r ont été affectées, il n'existe plus de contrainte partagée entre deux sous-problèmes différents.

Afin d'éviter d'exlorer plusieurs fois une même zone de l'espace de recherche, BTD enregistre des *(no)goods* structurels. Un good (resp. nogood) structurel est une affectation des variables d'un *séparateur* (*i.e.*, les variables qui appartiennent à l'intersection d'un cluster racine avec un de ses fils) tel que le sous-problème associé à ce fils est cohérent (resp. incohérent) quand on affecte ainsi ces variables. Ces (no)goods structurels permettent de réduire la complexité théorique de BTD à $\mathcal{O}(nd^{w+1})$ où n est le nombre de variables, d la taille du plus grand domaine, et w la largeur de T. La complexité en espace est $\mathcal{O}(nsd^s)$ où s est la taille du plus grand séparateur. Le nombre maximum de (no)goods structurels que BTD peut enregistrer sur un séparateur donné est borné par d^s.

3 Cadre générique d'une intégration de BTD dans une bibliothèque CP

Comme cela a été signalé dès son introduction [11], BTD peut utiliser n'importe quelle approche de résolution de CSP pour rechercher une solution dans un sous-problème défini par un cluster. Nous proposons ici un cadre générique pour l'intégration de BTD dans une bibliothèque CP, permettant ainsi de combiner BTD avec l'ensemble des techniques de résolution, algorithmes de propagation (qui n'altèrent pas la structure du problème en rajoutant des contraintes portant sur des variables qui ne se trouvent pas au sein d'un même cluster) et heuristiques intégrés dans la bibliothèque.

Notre cadre suppose uniquement que la bibliothèque CP fournit deux méthodes :

- une méthode *buildModel*$((X, D, C), K_i)$ qui retourne un modèle CP étant donnés une instance CSP (X, D, C) et un sous-ensemble de variables à instancier $K_i \subseteq X$;
- une méthode *next()* qui retourne une nouvelle fonction domaine $D' \subseteq D$ qui affecte de façon cohérente toutes les variables de K_i, ou bien retourne *null* s'il n'existe plus de solution.

La fonction générique qui intègre BTD dans une telle bibliothèque CP est décrite dans l'algorithme 1. Cette fonction prend en entrée une instance CSP (X, D, C), une décomposition arborescente (K, T), et un sommet racine r de T. *BTD* appelle tout d'abord la méthode *buildModel* afin de construire le modèle CP, où l'ensemble des variables à affecter est restreint à celles du cluster racine K_r. Ensuite, elle appelle itérativement la méthode *next* (ligne 2) pour rechercher des affectations cohérentes des variables de K_r, jusqu'à ce que soit une affectation puisse être étendue à une solution pour tous les sous-problèmes de r (de sorte que le problème est résolu - ligne 12), ou bien il n'existe plus d'autre affectation cohérente (de sorte que le problème est incohérent - ligne 13). Dans les lignes 3 à 10, *BTD* tente d'étendre l'affectation courante des variables de K_r à une solution pour tous les sous-problèmes : pour chaque fils i de r dans T, s'il existe un (no)good pour les valeurs affectées aux variables du séparateur $(K_r \cap K_i)$, alors ce sous-problème a déjà été résolu précédemment ; sinon *BTD* est récursivement appelée sur le sous-arbre de T enraciné en i, et le (no)good correspondant à ce sous-problème est enregistré. La boucle lignes 5 à 10 se termine soit quand un nogood est trouvé pour un fils (dans ce cas,

la recherche passe à l'affectation suivante des variables de K_r), soit quand il existe un good pour tous les fils (dans ce cas, la recherche s'arrête sur un succès, ligne 12).

Etant donné un modèle retourné par la méthode *buildModel*$((X, D, C), K_r)$, si la complexité en temps de l'ensemble des appels à la méthode *next()* (pour rechercher toutes les affectations cohérentes des variables de K_r) est $\mathcal{O}(|K_r|^d)$ où $d = \max_{x_i \in K_r} |D(x_i)|$, alors la complexité en temps de l'algorithme 1 est la même que celle de l'algorithme BTD introduit dans [11].

4 Intégration de politiques de restarts dans le cadre générique

Les bibliothèques CP récentes intègrent des politiques de restarts qui améliorent fortement les performances des méthodes de résolution de CSP. Une recherche avec restarts est composée de plusieurs recherches successives, chacune de ces recherches étant limitée (borne sur le temps ou sur le nombre d'échecs, par exemple) : quand la limite est atteinte, la recherche courante est arrêtée et une nouvelle recherche est lancée avec une nouvelle limite (typiquement, une limite plus grande). Chaque nouvelle recherche considère des heuristiques différentes, par exemple, de sorte que des zones différentes de l'espace de recherche sont explorées. Par ailleurs, des informations peuvent être transmises d'une recherche à l'autre, concernant par exemple l'ordre dans lequel affecter les variables ou bien des nogoods liés aux échecs [17].

L'intégration de restarts dans BTD peut se faire de deux manières orthogonales qui peuvent être combinées. La première qui a été proposée dans [13] consiste à changer de cluster racine à chaque tour. Cela conduit évidemment à modifier l'ordre d'affectation des variables et donc à explorer différentes parties de l'espace de recherche.

La seconde que nous explorons ici consiste à utiliser une méthode de résolution à base de restarts pour la résolution des clusters (recherche d'une solution dans le cluster courant). Plus précisément, quand nous demandons à la bibliothèque CP de rechercher une affectation cohérente des variables du cluster racine K_r (appel de la méthode *next()*, ligne 2 de l'algorithme 1), nous paramétrons la bibliothèque CP afin qu'elle utilise une politique de restarts. Cependant, les approches à base de restarts ne permettent pas directement de lister toutes les solutions d'un problème. En effet, rechercher une nouvelle solution juste après avoir trouvé une première solution peut conduire exactement à la même solution. Il est nécessaire donc de rajouter une contrainte (un nogood) interdisant la solution trouvée pour pouvoir en trouver potentiellement d'autres. Nous proposons de combiner ces nogoods avec des nogoods structurels (*i.e.*, des nogoods sur les séparateurs des clusters) : quand une affectation cohérente pour les variables du cluster racine K_r a été trouvée, BTD essaie récursivement de l'étendre à tous les sous-arbres de r ; s'il réussit (ligne 11), alors BTD s'arrête sur un succès et il n'est pas nécesssaire d'ajouter un nogood dans la mesure où le problème enraciné en r est résolu ; sinon, nous ajoutons au modèle Gecode le nogood correspondant aux valeurs affectées aux variables de $K_r \cap K_i$ où i est le fils de r pour lequel BTD a retourné un échec. Concrêtement, cela est réalisé en modifiant la ligne 10 de l'algorithme 1 pour ajouter le nogood A_i à *model*, en plus d'enregistrer A_i comme un nogood structurel.

Notons que les nogoods ajoutés au modèle Gecode sont des nogoods visant à empêcher la méthode *next()* de Gecode de retourner une solution qu'elle a déjà retournée précédemment. L'objectif est donc différent des nogoods introduits par Lecoutre *et al* dans [17], qui visent à éviter d'explorer plusieurs fois des zones de l'espace de recherche menant à des échecs. De fait, lorsque Gecode utilise une politique de restarts, ces nogoods liés aux échecs sont automatiquement gérés par Gecode.

Théorème 1 *La complexité en temps de l'algorithme 1 utilisant une politique de restarts est* $\mathcal{O}(nb_{restarts}.n^2d^{w+1+s})$ *avec* $nb_{restarts}$ *le nombre de restarts, n le nombre de variables, d la taille maximum des domaines, w la largeur de la décomposition arborescente et s la taille maximum des intersections entre clusters.*

Si une solution est trouvée sur un cluster, soit elle mène à une solution globale dans le sous-problème enraciné en ce cluster et celui-ci est résolu, soit elle ne mène pas à une solution globale dans ce sous-problème et un nogood est rajouté et on cherche la solution suivante. Sachant que la recherche de la solution suivante ne commence pas nécessairement là où s'était arrêté la dernière, son coût est en $O(d^{w+1})$. Cette recherche de la solution suivante est effectuée autant de fois que la solution précédente a conduit à un échec et donc à l'enregistrement d'un nogood. Le nombre maximum de nogoods qu'il est possible d'avoir est borné par $maxf.d^s$, avec $maxf$ le nombre maximum de fils du cluster courant. En outre, $maxf$ peut être borné par n. De ce fait, la complexité de cette version de BTD est en $O(n^2d^{w+1+s})$ pour chaque restart. Par ailleurs, nous avons noté un nombre de restarts limité sur les expérimentations que nous avons menées. Il est en moyenne de 4 par instance avec un maximum de 460 pour une instance très difficile contenant 680 variables.

5 Évaluation expérimentale

5.1 Description du benchmark et des conditions expériementales

Nous avons choisi d'évaluer les méthodes proposées sur les 3285 instances CSP du benchmark de la compétition CP'2009 (Fourth International CSP Solver Competition : http ://www.cril.univ-artois.fr/CSC09/). Ces instances ont été regroupées en cinq classes :

- la classe C1, qui contient 556 instances définies avec des contraintes globales (*alldiff, cumulative, element, weightedsum*),
- la classe C2, qui contient 709 instances avec des contraintes d'arité quelconque définies en intention,
- la classe C3, qui contient 686 instances avec des contraintes binaires définies en intention,
- la classe C4, qui contient 699 instances avec des contraintes d'arité quelconque définies en extension,
- la classe C5, qui contient 635 instances avec des contraintes binaires définies en extension.

Nous avons utilisé le modèle proposé dans [18] pour définir ces instances au sein du solveur Gecode. Toutes les expérimentations ont été menées sur des processeurs Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz, avec 20480 Ko de cache et 3Go de RAM. Chaque méthode a été exécutée sur ces instances avec une durée limite fixée à 30mn au-delà de laquelle cette instance est considérée comme non résolue par la méthode. Les temps reportés dans la suite de cette section incluent le temps nécessaire au prétraitement de l'instance et au calcul de la décomposition arborescente (détaillés en 5.2). Si la mémoire nécessaire pour la résolution d'une instance dépasse les 3Go de RAM, alors cette instance est également considérée comme non résolue par la méthode.

5.2 Approches considérées et détails d'implémentation

Nous avons implémenté l'algorithme 1 à l'aide de la bibliothèque Gecode [23], de façon très naturelle dans la mesure où Gecode fournit les deux méthodes *buildModel* et *next*, et permet d'ajouter des nogoods à un modèle. Nous appelons *BTD-G* (resp. *BTD-GR*) la variante utilisant Gecode sans restarts (resp. Gecode avec restarts) quand la méthode *next* est appelée.

Paramètres de Gecode. Une première série d'expérimentations nous a permis de constater que la configuration de Gecode (sans BTD) qui obtient les meilleurs résultats sur les instances considérées est celle qui utilise une politique de restarts combinée à l'heuristique de choix de variables INT_VAR_AFC_SIZE_MAX (correspondant à minDom/wdeg [4] avec un choix aléatoire en cas d'égalités) et un choix de valeurs aléatoire. La politique de restarts est basée sur une progression géométrique avec un facteur d'échelle fixé à 2000 et une base à 2. Les restarts sont combinés avec l'enregistrement d'un nogood à chaque fois qu'une affectation partielle incohérente est trouvée, la profondeur maximale d'enregistrement de ces nogoods étant limitée à sa valeur par défaut (*i.e.*, 128). Cette configuration de Gecode est appelée GecodeR dans la suite.

Nous utilisons les mêmes heuristiques de choix de variables et de valeurs que GecodeR pour BTD-G et BTD-GR, et la même politique de restarts pour BTD-GR.

Calcul d'une décomposition arborescente pour BTD-G et BTD-GR. BTD repose sur une décomposition arborescente d'un CSP, et sa complexité en temps dépend de la taille des clusters. Plus les clusters sont petits, plus le problème est simple à résoudre en théorie. Cependant, le calcul d'une décomposition minimisant la taille maximum des clusters est un problème NP-difficile. Ainsi, une heuristique est souvent utilisée pour calculer une triangulation de la 2-section de l'hypergraphe de contraintes (de sorte que tout cycle de longueur supérieure ou égale à 4 possède une arête reliant deux sommets non consécutifs). Cette propriété assure qu'on peut facilement calculer une décomposition arborescente du graphe dont les clusters sont ses cliques maximales. Dans notre cas, nous avons choisi d'utiliser l'approche consistant à construire d'abord un graphe dont les sommets représentent les cliques maximales et il existe une arête entre deux sommets si les cliques maximales correspondantes ont une intersection qui n'est pas vide. Ensuite, un arbre recouvrant est calculé depuis ce graphe. Cet arbre induit une décomposition arborescente. Nous utilisons l'heuristique de triangulation Minfill [16] qui est connue pour sa capacité à produire des clusters de taille réduite.

L'espace mémoire nécessaire au stockage des (no)goods structurels dépend de la taille maximale des séparateurs entre les clusters. Il est très important de limiter cette valeur [8]. Comme [8], nous avons fixé la borne maximum à ne pas dépasser à 10. Si deux clusters ont une intersection dont la taille dépasse cette valeur, ils sont fusionnés pour former un seul cluster.

Une fois que la décomposition arborescente a été calculée, nous devons choisir un cluster racine, à partir duquel la recherche BTD sera commencée. Ce cluster racine est choisi aléatoirement parmi l'ensemble des clusters qui ont le plus grand nombre de variables. Les fils de chaque cluster sont ordonnés par taille de sépa-

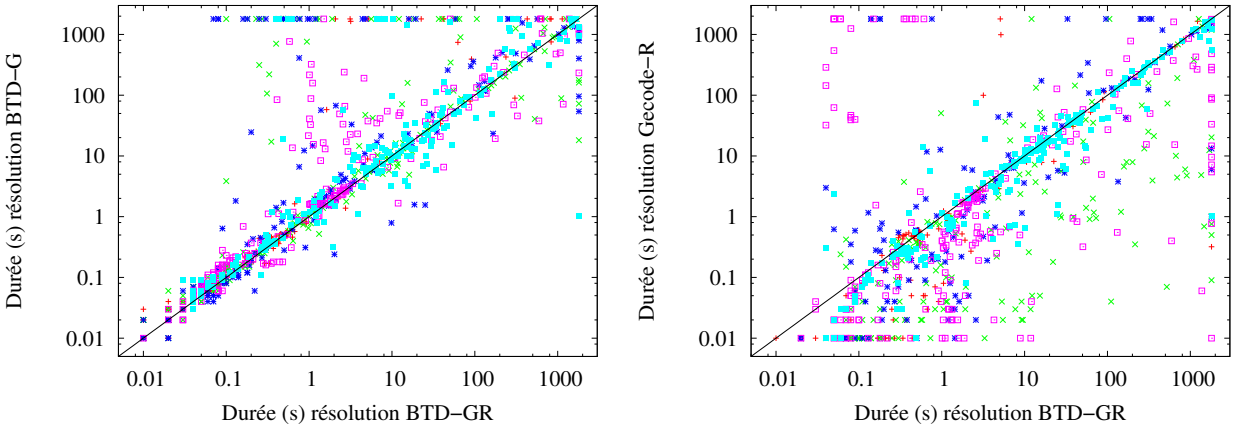


FIGURE 1 – Comparaison de BTD-G, BTD-GR et GecodeR. Chaque point (x, y) de la courbe de gauche (resp. de droite) correspond à une instance résolue en x secondes par BTD-GR, et y secondes par BTD-G (resp. GecodeR). La couleur et la forme des points dépendent de la classe de l'instance : $+$ pour C1, $\times$ pour C2, $*$ pour C3, $\square$ pour C4, et $\blacksquare$ pour C5.

rateur croissante : dans la boucle des lignes 5 à 10 de l'algorithme 1, nous choisissons en premier le fils i de r tel que $|K_i \cap K_r|$ soit minimal, puis le deuxième plus petit, etc.

Prétraitement sur les instances. Pour certaines instances, la 2-section n'est pas connexe. Dans ce cas, nous décomposons tout d'abord la 2-section en composantes connexes, et nous résolvons chaque sous-problème correspondant à chaque composante connexe séquentiellement. Cela est fait pour toutes les approches considérées (y compris GecodeR). Notons que le fait de résoudre chaque composante connexe séparément n'implique pas nécessairement que le sous-graphe de la 2-section induit par les variables d'un cluster soit nécessairement connexe, comme cela a été remarqué dans [13].

Pour chaque instance, nous demandons à Gecode de propager ses contraintes afin de filtrer ses domaines avant de commencer sa résolution. Gecode ne garantit pas que ce filtrage assure la cohérence d'arc généralisée pour toutes les contraintes : sur certaines, il peut être moins fort pour réduire le temps de calcul nécessaire. Néanmoins, ce filtrage suffit à résoudre 12 instances du benchmark. Par ailleurs, ce filtrage permet de réduire le domaine d'au moins une variable à une seule valeur pour 672 instances. Pour ces instances, nous éliminons de la 2-section les sommets correspondant à des variables dont le domaine est réduit à une seule valeur, avant de calculer la décomposition arborescente. Cela permet souvent d'avoir une meilleure décomposition en termes de taille des clusters : la largeur est diminuée pour 86% des 672 instances, et augmentée pour 6% de ces instances ; quand elle est diminuée, elle diminue de 37% (en moyenne, avec une diminution maximale de 98% pour une instance) ; quand elle est augmentée, elle augmente de 20% (en moyenne, avec une augmentation maximale de 31%).

5.3 Résultats expérimentaux

Parmi les 3285 instances du benchmark, 1880 ne sont pas structurées du tout dans le sens où leur décomposition arborescente ne contient qu'un seul cluster (contenant toutes les variables). Pour ces 1880 instances, BTD-G (resp. BTD-GR) se comportent comme Gecode sans restarts (resp. comme GecodeR). La seule différence est due au coût de calcul de la décomposition arborescente, qui est généralement très petit comparé au temps de résolution : en moyenne sur les 1880 instances non structurées, le coût de calcul de la décomposition arborescente est égal à 2% du temps CPU de résolution par GecodeR.

Parmi les 1405 instances pour lesquelles la décomposition arborescente comporte plus d'un cluster, 427 ne sont résolues par aucune des approches considérées dans la limite de temps CPU considérée (30 minutes). La figure 1 compare BTD-G avec BTD-GR (partie gauche), et BTD-GR avec GecodeR (partie droite) pour les 978 instances restantes (dont la décomposition comporte plus d'un cluster, et qui sont résolues par au moins une approche). La partie gauche nous montre que l'utilisation d'une politique de restarts pour chercher une affectation cohérente du cluster courant améliore la résolution pour un grand nombre d'instances, tandis que cela dégrade la résolution pour un nombre réduit d'instances. La partie droite nous montre que, si

	GecodeR					BTD-GR				
	C1	C2	C3	C4	C5	C1	C2	C3	C4	C5
1s	**4**	**13**	5	**46**	2	0	1	**6**	22	2
5s	1	**12**	2	12	**4**	1	1	**6**	**22**	2
10s	2	**12**	4	7	**5**	**3**	1	**7**	**21**	2
50s	1	**7**	3	9	**4**	**3**	4	**7**	**17**	1
100s	1	**6**	3	7	**3**	**3**	3	**7**	**16**	0
500s	1	**5**	3	5	0	**2**	2	**9**	**14**	0
1000s	1	2	3	3	0	1	2	**9**	**13**	0
1800s	1	**3**	3	0	**1**	1	2	**9**	**11**	0

TABLE 1 – Nombre d'instances bien structurées résolues par GecodeR et pas BTD-GR (partie gauche) et nombre d'instances bien structurées résolues par BTD-GR et non GecodeR (partie droite), pour différentes limites de temps CPU.

pour beaucoup d'instances la décomposition dégrade les performances, pour certaines instances elle l'améliore fortement. En particulier, 16 instances qui ne sont pas résolues par GecodeR dans la limite de temps de 1800 secondes sont résolues par BTD-GR en moins d'une seconde.

Regardons maintenant d'un peu plus près les instances pour lesquelles la décomposition arborescente est bonne, c'est-à-dire, les instances pour lesquelles le plus grand cluster de la décomposition arborescente comporte au plus cinquante pour cent des variables. Parmi les 1405 instances pour lesquelles la décomposition arborescente comporte plus d'un cluster, il y a 520 instances pour lesquelles la décomposition est bonne : 70 (resp. 126, 80, 103, and 141) instances de la classe C1 (resp. C2, C3, C4, et C5).

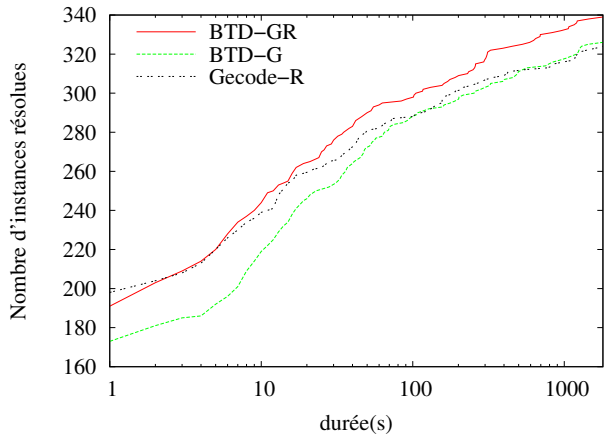


FIGURE 2 – Evolution du nombre d'instances bien structurées résolues par BTD-GR, BTD-G, et GecodeR en fonction du temps CPU.

La figure 2 nous montre que, pour ces instances bien structurées, BTD-GR est clairement plus performant que GecodeR, pour des limites de temps supérieures à 10 secondes, tandis que pour des limites de temps inférieures, les deux approches ont des résultats relativement similaires. Nous constatons également que BTD-GR est plus performant que BTD-G, ce qui confirme bien le fait que l'intégration d'une politique de restarts pour résoudre chaque cluster améliore le processus de résolution. Au bout de 1800 secondes, BTD-GR (resp. BTD-G et GecodeR) ont été capables de résoudre 339 (resp. 326 et 324) instances bien structurées.

La table 1 nous montre que chacune des deux approches BTD-GR et GecodeR est capable de résoudre des instances que l'autre n'est pas capable de résoudre. Il est intéressant de noter que les instances qui ne sont résolues que par BTD-GR appartiennent à toutes les classes, même si une majorité d'entre elles appartiennent à la classe C4 des instances n-aires définies en intention.

6 Conclusion

Nous avons montré dans cet article comment intégrer la méthode BTD dans une bibliothèque de programmation par contraintes, utilisée comme une boite noire pour résoudre des sous-problèmes. Cette intégration nous permet de bénéficier des algorithmes de résolution implémentés dans la bibliothèque (propagation de contraintes globales, heuristiques de choix de variables, restarts, etc). Nous avons réalisé cette intégration à l'aide de la bibliothèque Gecode. Les premières expérimentations sur le benchmark de la compétition CSP 2009 montrent tout le potentiel de cette intégration. En particulier, certaines instances difficiles pour la version considérée de Gecode sont bien mieux résolues en exploitant une décomposition arborescente.

Cette étude a également permis de montrer qu'une stratégie de restart peut être appliquée à l'intérieur de

chaque cluster, et que cette stratégie améliore les performances. Cette stratégie est complémentaire de celle proposée dans [13], et ces deux stratégies pourraient être combinées.

Cette première intégration de BTD dans une bibliothèque de contraintes nous ouvre de nombreuses perspectives de recherche. Dans la version actuelle, nous avons fait le choix de considérer toutes les contraintes initiales dans chaque cluster, comme cela est proposé dans [11, 13]. Cela permet à Gecode de filtrer au maximum les domaines, mais bien évidemment, cela peut être assez coûteux en temps. Une autre possibilité aurait été de restreindre l'ensemble des contraintes à celles dont la portée est incluse dans le cluster courant, ce qui est moins coûteux en temps mais peut également moins réduire les domaines des variables. Nous travaillons actuellement à l'extension de notre cadre d'intégration de BTD dans une bibliothèque de programmation par contraintes afin de pouvoir considérer différents sous-ensembles de contraintes, pour chaque sous-problème.

Références

[1] C. Bessière and J.-C. Régin. MAC and Combined Heuristics : Two Reasons to Forsake FC (and CBJ ?) on Hard Problems. In *Proceedings of CP*, pages 61–75, 1996.

[2] L. Blet, S. N. Ndiaye, and C. Solnon. Experimental comparison of BTD and intelligent backtracking : Towards an automatic per-instance algorithm selector. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP-2014)*, pages 190–206, 2014.

[3] Loïc Blet, Samba Ndojh Ndiaye, and Christine Solnon. A generic framework for solving csps integrating decomposition methods. In *CP doctoral program*, Quebec, Canada, 2012.

[4] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *ECAI-2004*, volume 16, page 146, 2004.

[5] C. P Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 431–437, 1998.

[6] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :343–282, 2000.

[7] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.*, 14(3) :263–313, 1980.

[8] P. Jégou, S. N. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP-2005)*, pages 777–781, 2005.

[9] P. Jégou, S. N. Ndiaye, and C. Terrioux. Dynamic Management of Heuristics for Solving Structured CSPs. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP-2007)*, pages 364–378, 2007.

[10] P. Jégou, S. N. Ndiaye, and C. Terrioux. Combined Strategies for Decomposition-based Methods for solving CSPs. In *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI 2009)*, pages 115–122, 2009.

[11] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146 :43–75, 2003.

[12] P. Jégou and C. Terrioux. Combining Restarts, Nogoods and Decompositions for Solving CSPs. In *Proceedings of the European Conference on Artifcial Intelligence (ECAI 2014)*, pages 465–470, 2014.

[13] P. Jégou and C. Terrioux. Tree-Decompositions with Connected Clusters for Solving Constraint Networks. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP-2014)*, pages 407–423, 2014.

[14] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artif. Intell.*, 139(1) :21–45, 2002.

[15] S. Karakashian, R. J. Woodward, and B. Y. Choueiry. Reformulating R(*, m)C with Tree Decomposition. In *Ninth International Symposium on Abstraction, Reformulation and Approximation (SARA 2011)*, pages 62–69, 2011.

[16] U. Kjaerulff. Triangulation of Graphs - Algorithms Giving Small Total State Space. Technical report, Judex R.R. Aalborg., Denmark, 1990.

[17] Christophe Lecoutre, Lakhdar Sais, SÃ©bastien Tabary, and Vincent Vidal. Recording and minimizing nogoods from restarts. *JSAT*, 1(3-4) :147–167, 2007.

[18] M. Morara, J. Mauro, and M. Gabbrielli. Solving xcsp problems by using gecode. In *26th Italian*

Conference on Computational Logic (CILC), volume 810 of *CEUR Workshop Proceedings*, pages 401–405. CEUR-WS.org, 2011.

[19] Cédric Pralet and Gérard Verfaillie. Travelling in the world of local searches in the space of partial assignments. In *CPAIOR*, pages 240–255, 2004.

[20] Patrick Prosser. Forward checking with backmarking. In *Constraint Processing, Selected Papers*, pages 185–204. Springer-Verlag, 1995.

[21] N. Robertson and P.D. Seymour. Graph minors II : Algorithmic aspects of tree-width. *Algorithms*, 7 :309–322, 1986.

[22] D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of the eleventh ECAI*, pages 125–129, 1994.

[23] C. Schulte, G. Tack, and M. Z. Lagerkvist. Modeling. In Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist, editors, *Modeling and Programming with Gecode*. 2013. Corresponds to Gecode 4.2.1.

Choix de nœud dans un algorithme de Branch and Bound sur intervalles

Bertrand Neveu[1] **Gilles Trombettoni**[2] **Ignacio Araya**[3 *]

[1] Imagine LIGM, Université Paris–Est, France
[2] LIRMM, Université de Montpellier, CNRS, France
[3] Pontificia Universidad Católica de Valparaíso, Chile
Bertrand.Neveu@enpc.fr,Gilles.Trombettoni@lirmm.fr,rilianx@gmail.com

Résumé

Nous présentons dans cet article de nouvelles stratégies de choix de nœud dans un algorithme de *Branch and Bound* sur intervalles pour l'optimisation globale sous contraintes. La stratégie standard en meilleur d'abord choisit le nœud qui a la plus petite borne inférieure de l'estimation de l'objectif dans le domaine correspondant. Nous proposons d'abord de nouvelles stratégies qui prennent aussi en compte la borne supérieure de l'estimation de l'objectif. Une bonne précision sur cette borne supérieure, obtenue par l'application de plusieurs opérateurs de contraction, permet d'obtenir plus rapidement de bonnes solutions réalisables et donc de meilleures performances. Nous proposons également une autre stratégie qui réalise un compromis entre diversification et intensification en plongeant de manière gloutonne dans des régions potentiellement réalisables à chaque nœud de la recherche en meilleur d'abord. Ces nouvelles stratégies obtiennent de meilleurs résultats expérimentaux que la recherche en meilleur d'abord sur des instances difficiles d'optimisation globale sous contraintes.

Abstract

We present in this article new strategies for selecting nodes in interval Branch and Bound algorithms for constrained global optimization. The standard best-first strategy selects a node with the lowest lower bound of the objective estimate. We first propose new node selection policies where an *upper bound* of each node/box is also taken into account. The good accuracy of this upper bound achieved by several contracting operators leads to a good performance of the criterion. We propose another strategy that also makes a tradeoff between diversification and intensification by greedily diving into potential feasible regions at each node of the best-first search. These new strategies obtain better experimental results than classical best-first search on difficult constrained global optimization instances.

*Ignacio Araya est financé en partie par le projet Fondecyt 11121366.

1 Introduction

Cet article traite d'optimisation globale continue déterministe calculée par un algorithme de *Branch and Bound* sur intervalles. Plusieurs travaux ont été réalisés pour trouver de bonnes stratégies de branchement [10], mais la sélection de nœud elle-même a été peu étudiée. Les solveurs sur intervalles suivent généralement une stratégie en meilleur d'abord (BFS), et quelques études ont été menées pour limiter la croissance exponentielle de la mémoire utilisée [6, 18].

Travaux connexes

À notre connaissance, Casado et al. dans [6] et Csendes dans [9] ont été les seuls à proposer des stratégies de choix de nœud pour des algorithmes de B&B sur intervalles. Un critère à maximiser appelé C_3 dans [14] et adapté ensuite pour l'optimisation globale sans contraintes est :

$$\frac{f^* - lb}{ub - lb}$$

où $[lb, ub] = [f]_N([x])$ est l'intervalle obtenu par l'évaluation naturelle par intervalles de la fonction objectif f sur la boîte courante $[x]$: $[lb, ub]$ est un intervalle incluant tous les nombres réels images d'un point de $[x]$ par f. f^* est l'optimum. Quand f^* n'est pas connu, $\widetilde{f}$, le coût du meilleur point réalisable trouvé jusque là, peut être utilisé comme approximation de f^*. Ce critère favorise les petites boîtes et les nœuds avec une valeur de lb faible.

Pour l'optimisation sous contraintes, un autre critère (à maximiser) nommé C_5 est égal à $C_3 \times fr$. Il prend en compte un *ratio de faisabilité* fr calculé à partir de toutes les contraintes d'inégalité. Le critère C_7 propose de minimiser $\frac{lb}{C_5}$.

D'autres stratégies de choix de nœud ont été étudiées pour des algorithmes de B&B (non sur intervalles) en profondeur d'abord, en largeur d'abord ou en meilleur d'abord. La recherche en profondeur d'abord favorise l'exploitation (l'intensification). La recherche en meilleur d'abord favorise plus l'exploration (la diversification) car deux noeuds successifs peuvent appartenir à des régions significativement différentes de l'espace de recherche.

Différentes stratégies sont proposées dans l'optimiseur SCIP [20]. La profondeur, la largeur et le meilleur d'abord sont disponibles. Avec l'option `restartdfs`, SCIP réalise une recherche en profondeur d'abord, mais périodiquement (c.-à-d. tous les $k = 100$ retours en arrière), choisit le meilleur nœud. La stratégie la plus sophistiquée, appelée *Upper Confidence bounds for Trees* (UCT) [13], est utilisée pour les problèmes MIP.

La sélection de nœud de l'optimiseur global Baron est brièvement décrite dans [21]. La stratégie par défaut alterne entre le nœud ayant la violation minimum et celui de plus petite borne inférieure de l'estimation du coût. La violation d'un nœud est définie comme la somme des erreurs pour chaque variable, ces erreurs étant calculées à partir de la solution de la relaxation convexe du problème. Quand la limitation en mémoire devient critique, Baron passe en profondeur d'abord.

Des stratégies de choix de nœud ont aussi été définies pour des problèmes MINLP convexes. La stratégie par défaut de l'optimiseur `Bonmin` [5] choisit le nœud avec la plus petite borne inférieure de l'estimation du coût. Les stratégies en profondeur et en largeur d'abord sont aussi disponibles. Finalement, une stratégie dynamique commence en profondeur d'abord et passe en largeur d'abord après avoir trouvé trois solutions entières réalisables.

La recherche K-Best-First Search [11] s'applique aux problèmes d'optimisation combinatoire. KBFS(k) réalise de manière itérative des cycles d'expansion de nœuds. A chaque cycle, les k meilleurs nœuds sont traités et leurs enfants sont mis dans la liste des nœuds à traiter. Ainsi, KBFS réalise un compromis entre des recherches en largeur et en meilleur d'abord. Si $k = 1$, KBFS(1) mène une recherche en meilleur d'abord. Si $k = \infty$, alors tous les nœuds à un niveau sont traités avant de passer au niveau suivant ; KBFS(∞) mène donc une recherche en largeur d'abord.

Contribution et plan

Cet article propose de nouvelles stratégies pour des algorithmes de B&B sur intervalles.

Après une présentation des bases nécessaires à la compréhension de l'article dans la partie 2, une première approche est décrite dans la partie 3. Cette approche utilise, non seulement une borne inférieure mais aussi une borne supérieure de l'estimation de l'objectif sur un nœud. Cette borne supérieure est rendue plus précise en utilisant des méthodes de contraction qui éliminent des régions infaisables. La stratégie que nous proposons alterne entre le nœud avec la plus petite borne inférieure et celui avec la plus petite borne supérieure.

Une seconde approche réalise simplement un compromis entre exploration et exploitation. Un algorithme de B&B par intervalle en meilleur d'abord exécute à chaque nœud une plongée gloutonne en profondeur d'abord pour se focaliser sur des régions faisables (voir partie 4).

Dans la partie 5, une étude empirique des différentes variantes de ces nouvelles stratégies montre qu'elles obtiennent de bonnes performances sur des problèmes difficiles d'optimisation non convexe appartenant au banc d'essai Coconut.

2 Algorithmes de Branch and Bound sur intervalles

Un intervalle $[x_i] = [\underline{x_i}, \overline{x_i}]$ définit l'ensemble des nombres réels x_i tels que $\underline{x_i} \leq x_i \leq \overline{x_i}$. Une *boîte* $[x]$ est un produit cartésien d'intervalles $[x_1] \times ... \times [x_i] \times ... \times [x_n]$. $w([x])$ représente la taille du plus grand intervalle de la boîte $[x]$.

Cet article traite d'optimisation globale continue sous contraintes d'inégalités définie par :

$$\min_{x \in [x]} f(x) \quad s.c. \quad g(x) \leq 0$$

où $f : \mathbb{R}^n \to \mathbb{R}$ est la fonction objectif (non convexe) et $g : \mathbb{R}^n \to \mathbb{R}^m$ est un un vecteur de fonctions (non convexes) [1] $x = \{x_1, ..., x_i, ...x_n\}$ est un vecteur de variables dont le domaine est la boîte $[x]$. x est dit *réalisable* s'il satisfait les contraintes.

Un schéma de B&B sur intervalles pour l'optimisation globale continue sous contraintes (aussi connue comme programmation non linéaire NLP) est décrit ci-après. Les algorithmes 1 et 2 correspondent aux algorithmes implantés dans IbexOpt [22] et IBBA [19] et permettent de comprendre les stratégies de choix de nœud qui suivent. Par ailleurs, ces stratégies peuvent être incorporées dans tout algorithme de branch and bound sur intervalles.

L'algorithme de B&B maintient deux types principaux d'information durant la recherche : $\tilde{f}$, le coût du meilleur point réalisable trouvé jusque là et f_{min} la plus petite borne inférieure $[x].lb$ de l'estimation de l'objectif sur les nœuds $[x]$ à explorer. En d'autres termes, pour chaque boîte $[x]$, il est garanti qu'il n'existe pas de point réalisable avec un coût inférieur à $[x].lb$.

1. Les équations $h_j(x) = 0$ peuvent être relâchées en deux inégalités $-\epsilon \leq h_j(x) \leq +\epsilon$.

Algorithm `IntervalBranch&Bound` *(f, g, x, box, ϵ_{obj}, ϵ_{sol})*

$f_{min} \leftarrow -\infty$; $\widetilde{f} \leftarrow +\infty$; $x_{\widetilde{f}} \leftarrow \bot$
$f^{sb}_{min} \leftarrow +\infty$ /* La plus petite borne inférieure des nœuds ayant atteint la précision ϵ_{sol}. */
$Boxes \leftarrow \{box\}$
while $Boxes \neq \emptyset$ **and** $\widetilde{f} - f_{min} > \epsilon_{obj}$ **and** $\frac{\widetilde{f}-f_{min}}{|\widetilde{f}|} > \epsilon_{obj}$ **do**
  $[x] \leftarrow$ `BestBox` ($Boxes$, criterion) ; $Boxes \leftarrow Boxes \setminus \{[x]\}$
  $([x]_1, [x]_2) \leftarrow$ `Bisect` ($[x]$)
  $([x]_1, x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` ($[x]_1$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$)
  $([x]_2, x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` ($[x]_2$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$)
  $(f^{sb}_{min}, Boxes) \leftarrow$ `UpdateBoxes` ($[x]_1$,ϵ_{sol}, f^{sb}_{min},$Boxes$)
  $(f^{sb}_{min}, Boxes) \leftarrow$ `UpdateBoxes` ($[x]_2$,ϵ_{sol}, f^{sb}_{min}, $Boxes$)
  $f_{min} \leftarrow \min(f^{sb}_{min}, \min_{[x]\in Boxes}\ [x].lb)$

Algorithm 1: Branch and Bound sur intervalles

L'algorithme est lancé avec l'ensemble de contraintes g, la fonction objectif f et avec la boîte des domaines d'entrée initialisant la liste $Boxes$ des boîtes à traiter. ϵ_{obj} est la précision requise sur l'objectif et est utilisé comme critère d'arrêt. Nous ajoutons au système une variable x_{obj} correspondant à l'image (au coût) de la fonction objectif et une contrainte $f(x) = x_{obj}$.

La procédure `BestBox` choisit le prochain nœud à traiter. Si le critère choisit un nœud $[x]$ avec la plus petite borne inférieure de l'estimation de l'objectif ($[x].lb$), l'algorithme B&B suivra la stratégie la plus commune (le meilleur d'abord).

La boîte choisie est ensuite coupée en deux sous-boîtes $[x]_1$ et $[x]_2$ le long d'une dimension choisie par une stratégie de branchement.

Les deux sous-boîtes sont alors traitées par la procédure `Contract&Bound` (voir l'algorithme 2). Une contrainte $x_{obj} \leq \widetilde{f} - \epsilon_{obj}$ est ajoutée au système pour diminuer la borne supérieure de l'estimation de l'objectif dans la boîte. La valeur ϵ_{obj} permet que la prochaine solution trouvée soit significativement meilleure que le meilleur point réalisable courant. La procédure `Contraction` *contracte* ensuite la boîte courante sans perdre de partie réalisable. En d'autres termes, quelques parties non réalisables aux bords du domaine sont enlevées par des algorithmes issus de la programmation par contraintes (CP) et des algorithmes de convexification. Cette contraction travaille sur la boîte étendue incluant la variable objectif x_{obj}. Ainsi, augmenter $\underline{x_{obj}}$ revient à augmenter la borne inférieure de l'objectif sur la boîte courante.

La procédure `FeasibleSearch` appelle ensuite une ou plusieurs heuristiques recherchant un point réalisable $x_{\widetilde{f}}$ qui améliore le meilleur coût trouvé jusqu'à présent (*upperbounding*).

Les derniers appels à `UpdateBoxes`, décrits à l'algorithme 3, mettent généralement les deux sous-boîtes dans l'ensemble $Boxes$ des nœuds à explorer. Si la taille d'une boîte atteint la précision ϵ_{sol}, cette boîte ne sera pas étudiée (bissectée) et seule la valeur f^{sb}_{min} sera mise à jour.

Algorithm `Contract&Bound` *($[x]$,f,g,x,ϵ_{obj},$x_{\widetilde{f}}$,$\widetilde{f}$)*

$g' \leftarrow g \cup \{x_{obj} \leq \widetilde{f} - \epsilon_{obj}\}$
$[x] \leftarrow$ `Contraction` ($[x]$, $g' \cup \{f(x) = x_{obj}\}$)
if $[x] \neq \emptyset$ **then**
  /* Upperbounding */
  $(x_{\widetilde{f}}$,cost)$\leftarrow$`FeasibleSearch` ($[x]$,f,g',ϵ_{obj})
  if $cost < \widetilde{f}$ **then** $\widetilde{f} \leftarrow$ cost
return ($[x]$, $x_{\widetilde{f}}$, $\widetilde{f}$)

Algorithm 2: La procédure Contract&Bound lancée à chaque nœud de l'algorithme B&B

On remarquera que si l'arbre de recherche est parcouru en meilleur d'abord, une mémoire de taille exponentielle peut être requise pour stocker les nœuds à traiter.

Rappelons que $[x].lb$ est un minorant de l'objectif dans la boîte $[x]$ en tenant compte des contraintes (il n'existe aucun point réalisable avec un coût inférieur à $[x].lb$) et $[x].ub$ est un majorant de l'objectif pour le prochain point réalisable recherché dans la boîte courante.

L'arithmétique par intervalles appliquée sur l'objectif donne facilement des premières valeurs pour ces bornes. Avec l'arithmétique par intervalles, on remplace les opérateurs mathématiques standard dans f par leurs équivalents sur intervalles pour calculer $[f]([x])$. Ainsi, $\underline{[f]([x])}$ donne une valeur pour $[x].lb$ tandis que $\overline{[f]([x])}$ calcule une valeur pour $[x].ub$. Nous

Algorithm `UpdateBoxes` *([x], ϵ_{sol}, f^{sb}_{min}, Boxes)*
  if $[x] \neq \emptyset$ **then**
    if $w([x]) > \epsilon_{sol}$ **then**
      `Push` *([x], Boxes)*
    else
      $f^{sb}_{min} \leftarrow min(f^{sb}_{min}, \underline{x_{obj}})$
      /* or $f^{sb}_{min} \leftarrow min(f^{sb}_{min}, \underline{[f]([x])})$ */
  return $(f^{sb}_{min}$, *Boxes*$)$

Algorithm 3: Procédure de mise à jour des listes de nœuds à explorer ou du coût minimum de l'objectif sur les petites boîtes qui ne seront pas étudiées

verrons dans la section suivante comment ces bornes sont affinées en tenant compte des contraintes, donc de l'espace réalisable, et de la précision ϵ_{obj}.

3 Nouvelles stratégies basées sur le calcul de bornes supérieures

En optimisation, le choix du prochain nœud à traiter est crucial pour obtenir de bonnes performances. Si la stratégie habituelle de choix du nœud ayant la plus petite borne inférieure de l'estimation de l'objectif (stratégie en meilleur d'abord) permet une recherche optimale en optimisation sans contraintes, ce n'est plus le cas avec des contraintes.

Il est aussi intéressant d'obtenir rapidement de bons points réalisables, pour limiter d'une part la taille de la liste des nœuds à traiter et pour réduire l'espace de recherche en propageant les contraintes. Le meilleur nœud est alors celui qui améliore le plus le meilleur point réalisable. En effet, cette amélioration de la borne supérieure réduit *globalement* l'espace réalisable grâce à la contrainte $x_{obj} \leq \widetilde{f}$. Il existe deux phases dans un B&B :

1. une phase où on essaye de trouver la solution optimale,
2. une seconde phase où on prouve que cette solution est optimale, ce qui requiert de traiter tous les nœuds restants.

Le choix de nœud ne concerne donc que la première phase.

Nous définissons dans cette section de nouvelles stratégies agrégeant deux critères pour choisir la boîte courante :

1. LB : Le critère bien connu utilisé en meilleur d'abord choissant le nœud ayant le plus petit $[x].lb$. Ce critère est optimiste puisque nous espérons trouver une solution avec un coût f_{min}, auquel cas la recherche est terminée.
2. UB : Ce critère choisit le nœud ayant la plus petite borne supérieure de l'estimation de l'objectif, le nœud avec le plus petit $[x].ub$. Ainsi, si un point réalisable est trouvé, on espère qu'il améliorera de manière importante le meilleur coût trouvé jusque là.

Le critère UB est le symétrique du critère LB. Pour chaque boîte, $[x].lb$ et $[x].ub$ sont calculés par la procédure `Contract&Bound` et étiquettent le nœud avant de le stocker dans l'ensemble des nœuds à traiter. Rappelons qu'une variable x_{obj} représentant la valeur de l'objectif a été introduite. Ainsi, $[x].lb$ et $[x].ub$ sont simplement les bornes de l'intervalle $[x_{obj}]$ après contraction. Des techniques de programmation par contraintes comme 3BCID [23] peuvent améliorer $[x_{obj}]$ en traitant et enlevant de petites tranches aux bornes de $[x_{obj}]$ (rognage).

Nous pensons qu'une clé du succès du critère UB est qu'il évalue l'objectif plus précisément que l'évaluation naturelle ne le ferait (ub en calculant $[lb, ub] \leftarrow [f]([x])$).

Nous proposons deux manières d'agréger ces deux critères.

Sommer les deux critères : `LB+UB`. Cette stratégie choisit le nœud avec la plus petite valeur de la somme $[x].lb+[x].ub$. Cela correspond à minimiser les deux critères avec le même poids, c.-à-d. minimiser le milieu de l'intervalle représentant l'estimation de l'objectif dans la boîte.

Alterner les deux critères : `LBvUB`. Dans cette deuxième stratégie, on choisit la prochaine boîte à traiter en utilisant *un* des deux critères. Un choix aléatoire est effectué par la fonction *BestBox* à chaque selection de nœud, avec une probabilité `UBProb` de choisir UB. Si UB (resp. LB) est choisi et si plusieurs nœuds ont le même coût $[x].ub$ (resp. $[x].lb$), alors nous utilisons l'autre critère LB (resp. UB) pour départager les ex-aequo.

Des expérimentations ont montré que les performances n'étaient pas sensibles à un réglage fin du paramètre `UBProb` pourvu qu'il reste entre 0.2 et 0.8 ; ainsi il a été fixé à 0.5. Les expérimentations de la partie 5 soulignent l'impact positif de ce critère sur les performances.

3.1 Améliorer le critère UB en favorisant les régions réalisables

Tous les nœuds à explorer ont une étiquette UB dépendant du meilleur coût trouvé au moment où les boîtes ont été traitées par `Contract&Bound`. Grâce aux modifications apportées à `Contract&Bound` (voir l'algorithme 4), ces étiquettes tombent dans l'une des quatre catégories de coût dépendant de la valeur $\widetilde{f}$.

Algorithm `Contract&Bound` $([x], f, g, x, \epsilon_{obj}, x_{\widetilde{f}}, \widetilde{f})$

$g' \leftarrow g \cup \{x_{obj} \leq \widetilde{f} - (\epsilon_{obj} - 0.1\epsilon_{obj})\}$
$[x] \leftarrow$ `Contraction` $([x], g' \cup \{f(x) = x_{obj}\})$
if $[x] \neq \emptyset$ **then**
  // Upperbounding :
  $(x_{\widetilde{f}}, \text{cost}) \leftarrow$ `FeasibleSearch` $([x], f, g', \epsilon_{obj})$
  if $cost < \widetilde{f}$ **then**
    $\widetilde{f} \leftarrow \text{cost}$
    $[x].ub \leftarrow \widetilde{f} - \epsilon_{obj}$
  else
    $[x].ub \leftarrow \overline{x_{obj}}$
return $([x], x_{\widetilde{f}}, \widetilde{f})$

Algorithm 4: Modification de la procédure `Contract&Bound` pour améliorer le critère UB

Elles indiquent avec quelle priorité les boîtes sont choisies quand le critère UB est utilisé. L'étiquette UB $[x].ub$ est :

1. inférieure à $\widetilde{f} - \epsilon_{obj}$ si la procédure de contraction a réduit le maximum de l'estimation de l'objectif dans la boîte,
2. égale à $\widetilde{f} - \epsilon_{obj}$, si la boîte est un descendant de la boîte contenant le meilleur point réalisable actuel de coût $\widetilde{f}$,
3. égale à $\widetilde{f} - 0.9\,\epsilon_{obj}$ si la boîte a été traitée après la dernière mise à jour de $\widetilde{f}$,
4. supérieure à $\widetilde{f} - 0.9\,\epsilon_{obj}$ dans le cas restant.

Comme on le voit dans le pseudo-code de `Contract&Bound`, le terme additionnel $0.1\,\epsilon_{obj}$ pénalise les boîtes qui ne sont pas issues par bissections successives de la boîte où le meilleur point réalisable actuel a été trouvé.

Les bons résultats expérimentaux obtenus par la stratégie `LBvUB` (cf. partie 5) suggèrent qu'il est important de réaliser à la fois une intensification (UB) et une diversification (LB) de la recherche. L'ajout d'un second critère permet d'éviter les inconvénients de l'utilisation d'un seul critère, c.-à-d. (pour LB) choisir des boîtes prometteuses sans point réalisable et (pour UB modifié) aller plus en profondeur dans l'arbre de recherche en restant dans un minimum local.

3.2 Implantation de l'ensemble des nœuds à explorer

Dans `IbexOpt`, l'ensemble *Boxes* des nœuds à explorer était initialement implanté par une structure de tas ordonné par le critère LB. Pour la stratégie `LBvUB`, chaque nœud dans l'ensemble *Boxes* est étiqueté par deux valeurs : $[x].lb$ et $[x].ub$ et on a essayé plusieurs implantations pour trier les nœuds selon les deux critères LB et UB.

1. *Un tas* ordonné par $[x].lb$:
   Le choix de nœud utilisant UB est alors effectué en un coût linéaire par rapport au nombre de nœuds à explorer. En pratique, le temps nécessaire à la gestion du tas représente environ 10% du temps de calcul total quand ce nombre de nœuds dépasse 50000.
2. Une variante utilisant toujours un seul tas teste la taille du tas $|Boxes|$:
   Si $|Boxes|$ dépasse 50000, la probabilité `UBProb` est mise à 0.1 ; sinon, `UBProb` reste égal à 0.5. Cette variante produit donc moins d'appels avec le critère UB.
3. *Deux tas* :
   Finalement, nous avons essayé une implantation avec deux tas, l'un pour LB, l'autre pour UB. Toutes les opérations sont alors en $O(log_2(|Boxes|))$, sauf le processus de filtrage du tas lancé quand un nouveau point réalisable est trouvé. Ce ramasse-miettes enlève de *Boxes* tous les nœuds ayant $[x].lb > \widetilde{f}$.

4 La stratégie "Feasible Diving"

Nous avons conçu une autre stratégie de choix de nœud qui réalise aussi un compromis entre exploitation et exploration. L'algorithme B&B par intervalles réalise une recherche en meilleur d'abord décrite à l'algorithme 5.

Cette stratégie utilise une variante du critère LB : elle choisit un nœud avec le plus petit $[x].lb$, mais nous avons juste ajouté deux autres critères pour départager les ex-aequo :

1. le nœud le plus haut dans l'arbre de recherche.
2. en cas d'égalité sur $[x].lb$ et sur ce critère, on choisit le nœud généré en premier.

À chaque nœud de cette recherche, nous appelons une procédure effectuant une recherche gloutonne en profondeur d'abord pour intensifier la recherche dans la région de la boîte courante. Cette procédure *Feasible Diving* est décrite à l'algorithme 6. À partir du nœud sélectionné, `FeasibleDiving` construit un arbre en profondeur et garde le nœud le plus prometteur à chaque itération. La boîte est bissectée sur une dimension choisie par la même stratégie que dans l'algorithme B&B, par exemple *SmearSum relative* [22] et les deux sous-boîtes sont traitées par la procédure `Contract&Bound` (voir l'algorithme 2). La sous-boîte avec la plus petite borne inférieure de l'objectif est traitée par l'itération suivante de *Feasible Diving* tandis que l'autre est mise dans le tas global *Boxes* stockant la liste des nœuds à explorer.

Algorithm `IntervalBranch&BoundBis` *(f, g, x, box, ϵ_{obj}, ϵ_{sol})*

$f_{min} \leftarrow -\infty$; $\widetilde{f} \leftarrow +\infty$; $x_{\widetilde{f}} \leftarrow \bot$
$f^{sb}_{min} \leftarrow +\infty$ /* La plus petite borne inférieure des nœuds ayant atteint la précision ϵ_{sol}. */
$Boxes \leftarrow \{box\}$
while $Boxes \neq \emptyset$ **and** $\widetilde{f} - f_{min} > \epsilon_{obj}$ **and** $\frac{\widetilde{f}-f_{min}}{|\widetilde{f}|} > \epsilon_{obj}$ **do**
  $[x] \leftarrow$ BestBox ($Boxes$, criterion) ; $Boxes \leftarrow Boxes \setminus \{[x]\}$
  $([x], x_{\widetilde{f}}, \widetilde{f}, f^{sb}_{min}, Boxes) \leftarrow$ `FeasibleDiving` ($[x]$, g, f, x, ϵ_{sol}, $x_{\widetilde{f}}$, $\widetilde{f}$, f^{sb}_{min}, $Boxes$)
  if $[x] \neq \emptyset$ /* $w([x]) < \epsilon_{sol}$ */ **then** $f^{sb}_{min} \leftarrow min(f^{sb}_{min}, \underline{x_{obj}})$
  $f_{min} \leftarrow \min(f^{sb}_{min}, \min_{[x]\in Boxes} [x].lb)$

Algorithm 5: Branch and Bound sur intervalles (variante appelant Feasible Diving)

Algorithm `FeasibleDiving` *($[x]$, g, f, x, ϵ_{sol}, $x_{\widetilde{f}}$, $\widetilde{f}$, f^{sb}_{min}, $Boxes$)*

while $[x] \neq \emptyset$ **and** $w([x]) > \epsilon_{sol}$ **do**
  $([x]_1, [x]_2) \leftarrow$ `Bisect` ($[x]$)
  $([x]_1, x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` ($[x]_1$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$)
  $([x]_2, x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` ($[x]_2$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$)
  $([x]_{best}, [x]_{worst}) \leftarrow$ `Sort` ($[x]_1$, $[x]_2$, *criterion*) /* $[x]_{best}.lb < [x]_{worst}.lb$ */
  $(f^{sb}_{min}, Boxes) \leftarrow$ `UpdateBoxes` ($[x]_{worst}$,ϵ_{sol}, f^{sb}_{min}, $Boxes$)
  $[x] \leftarrow [x]_{best}$
return ($[x]$, $x_{\widetilde{f}}$, $\widetilde{f}$, f^{sb}_{min}, $Boxes$)

Algorithm 6: La procédure `Feasible Diving` lancée à chaque nœud du B&B en meilleur d'abord.

La procédure `FeasibleDiving` essaye de plonger dans une région réalisable (ou de montrer que la région explorée ne contient pas de point réalisable) grâce aux procédures `Contraction` et `FeasibleSearch` appelée à chaque itération. Avec cette plongée, on espère que ces procédures trouveront des points réalisables dans des petites sous-boîtes.

Il est mentionné dans [5] et [4] qu'une procédure similaire a été proposée pour des solveurs MIP. Bien que non détaillée dans la littérature, cette procédure *Probed Diving* semble être utilisée par le solveur MIP CPLEX d'IBM Ilog avec un certain succès.

5 Expérimentations

Nous avons mené des expérimentations sur un ensemble de problèmes d'optimisation globale continue sous contraintes. La partie 5.1 décrit le protocole expérimental. La partie 5.2 détaille les essais réalisés avec les différentes stratégies mixant LB et UB, spécialement `LB+UB` et `LBvUB`. Les parties 5.3 et 5.4 comparent `FeasibleDiving` à la stratégie standard en meilleur d'abord et à `LBvUB`.

5.1 Protocole expérimental et implantation

Toutes les variantes de notre solveur NLP `IbexOpt` [22] ont été implantées dans la bibliothèque libre en `C++` Interval-Based EXplorer (`Ibex`) [8, 7]. Les principaux composants de `IbexOpt` sont :

- Pour la contraction et l'estimation de la borne inférieure :
  - des contracteurs provenant de la programmation par contraintes sur intervalles comme l'algorithme bien connu de propagation de contraintes HC4 [15, 3] et le contracteur plus récent ACID [17].
  - des contracteurs réalisant une relaxation linéaire des contraintes et de l'objectif `X-Taylor` [1] et `ART` [16] (relaxation basée sur l'arithmétique affine).
- Pour la recherche d'un point réalisable :
  Une approche originale extrait une *région intérieure* (boîte ou polytope) dans l'espace réalisable en utilisant les algorithmes `InHC4` et `InXTaylor` [2].

On notera qu'aucune méthode lagrangienne ou d'analyse convexe n'est utilisée dans la version courante d'`IbexOpt`.

Nous avons sélectionné toutes les instances des séries 1 et 2 du banc d'essai sur l'optimisation globale sous contraintes Coconut [2] qui :

— sont résolues en un temps compris entre 1 seconde et 1 heure par un compétiteur (précision ϵ_{obj} = `1e-8`),
— possèdent entre 6 et 50 variables,
— sont résolues par `IbexOpt` en utilisant la meilleure stratégie de branchement parmi *Smear sum* absolue (ssa) ou relative (ssr), *Smear max* (sm), *Intervalle le plus large* (lf) et *Tour de rôle* (rr).

Cela donne les 84 instances listées en annexe (voir tableau 6).

5.2 Tests sur des stratégies mixtes LB/UB

Les expérimentations sur les meilleures stratégies mélangeant les critères LB et UB sont décrites dans cette partie. Par rapport à la stratégie LB standard (en meilleur d'abord), la meilleure stratégie de choix de nœud obtient un gain d'environ 40% sur le temps total et de 25% en moyenne, ce qui signifie que les plus grands gains sont obtenus sur les problèmes difficiles. De plus, on obtient un gain significatif sur plusieurs instances et la stratégie est robuste, c.-à-d. aucune perte importante par rapport à LB n'a été observée.

Comparaison entre LB+UB et les variantes LBvUB

Le tableau 1 montre les résultats expérimentaux obtenus avec différentes variantes mixant les critères LB et UB. Les gains/pertes par rapport à LB sont exprimés par $\frac{time(LB)}{time(LB/UB)}$.

Critère	#tas	t(s) max perte	t(s) max gain	t(s) moy. gain	t(s) total gain	nds moy. gain	nds total gain
LB+UB	1	0.14	10.2	1.21	1.54	1.28	1.70
LBvUB	1	0.52	9.17	1.17	1.58	1.20	1.75
LBvUB-01	1	0.52	21.7	1.18	1.64	1.19	1.67
LBvUB	2	0.46	21.7	1.28	1.67	1.26	1.83

TABLE 1 – Comparaison entre `LB+UB` et les variantes `LBvUB`. `LBvUB-01` est `LBvUB` avec la probabilité *UBProb* égale à 0.1 quand la taille du tas dépasse 50000 nœuds.

D'après ces expérimentations, LBvUB avec deux tas (un pour chaque critère) semble être la meilleure variante et l'expérimentation suivante justifie le choix de cette variante pour le critère UB.

2. `www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`

Quel est le meilleur critère UB ?

Nous avons réalisé d'autres tests sur ces 84 instances avec une précision sur l'objectif ϵ_{obj} = `1.e-6` (au lieu de `1.e-8`) en utilisant la version plus récente 2.1.10 d'`Ibex`. Toutes ces expérimentations utilisent une stratégie "LBvAutre" (avec une probabilité 0.5 de choisir un des deux critères pour le choix de nœud). Le premier critère est LB et l' "autre" critère est :

— UB_0 : UB sans biais favorisant les descendants (sous-boîtes) dans l'arbre du B&B du nœud où le dernier $\widetilde{f}$ a été trouvé.
— UB_1 : UB avec la modification de la procédure `Contract&Bound` décrite dans l'algorithme 4 qui favorise les descendants du nœud où le dernier $\widetilde{f}$ a été trouvé.
— FUB : UB avec un biais plus fort pour les descendants des nœuds où un point réalisable a été trouvé. Un tel descendant avec le plus petit UB est choisi, si il existe ; dans le cas contraire, on choisit un nœud avec le plus petit UB.
— MID : valeur de la fonction objectif au milieu de la boîte, donc en oubliant la région réalisable.
— C_3, C_5 ou C_7 : les critères décrits dans [14] et mentionnés dans l'introduction.

Le tableau 2 montre une comparaison synthétique de toutes ces variantes. Nous utilisons une formule plus sophistiquée pour les gains, qui donne une mesure plus équitable. La valeur dans chaque case correspond à une moyenne d'un ratio de temps normalisé entre une stratégie s et la stratégie de référence LB. Une valeur inférieure à 1 correspond à un gain, une valeur supérieure à 1 une perte. La mesure prend en compte le temps CPU d'une stratégie s pour résoudre une instance i (ce temps CPU est en fait une moyenne sur 10 essais effectués avec différentes graines pour les choix aléatoires effectué dans IbexOpt). Il est calculé de la manière suivante :

— $timeRatio(i,s) = \frac{time(i,s)}{time(i,s)+time(LB,i)}$
— $averageTimeRatio(s)$ est la moyenne des $timeRatio(i,s)$.
— ratio de temps normalisé (valeur pour les gains de s) = $\frac{averageTimeRatio(s)}{1-averageTimeRatio(s)}$

	UB_0	UB_1	FUB	MID	C_3	C_5	C_7
sur les 84 instances	0.87	0.84	0.87	0.91	0.93	1.03	1.08
sur les inst. > 10 s.	0.80	0.74	0.77	0.88	0.87	0.94	0.99

TABLE 2 – Comparaison entre les stratégies de choix de nœud `LB` et `LBvAutre` (`Autre` définissant la colonne)

La stratégie `LBvUB`, avec la variante UB_1 présentée à l'algorithme 4, obtient les meilleurs résultats. On observe un gain de 26% de UB_1 sur LB sur les problèmes difficiles.

Les critères C_3, C_5 et C_7 ont donné de moins bons résultats sur notre échantillon. Cela suggère qu'une évaluation précise de $[x].lb$ et de $[x].ub$ obtenue par contraction est en pratique meilleure qu'une évaluation par l'arithmétique d'intervalles de ces valeurs et que le *ratio de faisabilité* n'est peut être pas une mesure pertinente.

Comparaison détaillée entre LB et LBvUB

La figure 1 montre un diagramme comparant la meilleure variante de `LBvUB` ($\texttt{LBvUB}_1$) avec LB. Le tableau 3 confirme les bénéfices de LBvUB.

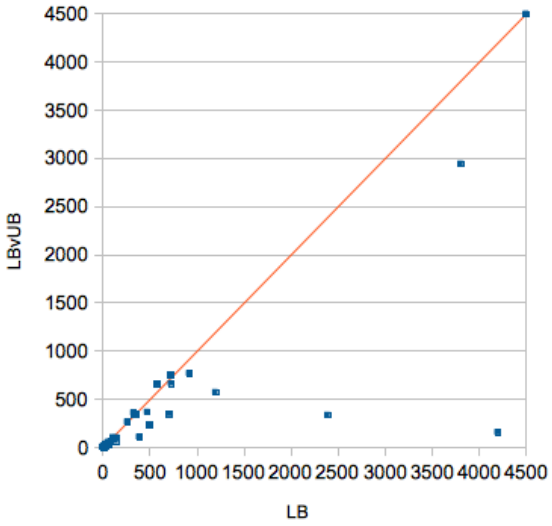


FIGURE 1 – `LB` versus `LBvUB`. Les coordonnées de chaque point représentent le temps CPU (en secondes) utilisé par les compétiteurs. La plupart des points sont sous la diagonale, ce qui illustre le fait que LBvUB domine LB.

5.3 Comparison entre LB et FeasibleDiving

La figure 1 montre un diagramme comparant `FeasibleDiving` avec LB. Le tableau 3 confirme les bénéfices de `FeasibleDiving`.

5.4 Comparaison entre LBvUB et FeasibleDiving

Nous concluons cette partie expérimentale en comparant les deux stratégies présentées dans cet article.

La figure 3 montre un diagramme comparant la meilleure variante de `LBvUB` ($\texttt{LBvUB}_1$) à FD. Le tableau 5 confirme les bénéfices de `FeasibleDiving` qui permet de résoudre les deux instances `concon` et `mconcon`.

Remarque : rappelons que l'algorithme B&B appelle une procédure de recherche de meilleur point réalisable à chaque nœud exploré par `FeasibleDiving`.

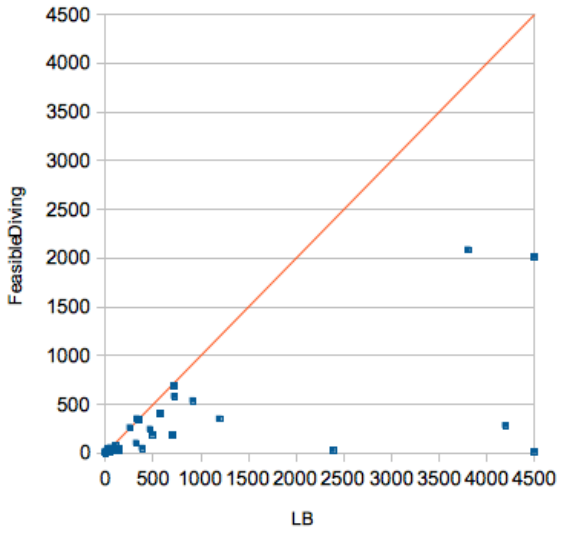


FIGURE 2 – LB versus FD. Les coordonnées de chaque point représentent le temps CPU (en secondes) utilisés par les compétiteurs. La plupart des points apparaissent sous la diagonale ce qui montre que FD domine nettement LB.

En effet, chaque itération de `FeasibleDiving` appelle `Contract&Bound` qui lui-même appelle les procédures `Contraction` et `FeasibleSearch`. Il est possible d'appeler si souvent `FeasibleSearch` dans `IbexOpt` parce que cela correspond à deux procédures relativement peu coûteuses (`InHC4` et `In-XTaylor` dans [2]). Quand l'algorithme plonge, la boîte courante devient petite et est davantage susceptible soit de contenir un espace réalisable que ces procédures arrivent à trouver, soit d'être éliminée par les procédures de contraction. Ceci pourrait expliquer les bons résultats obtenus par `FeasibleDiving`.

6 Conclusion

La stratégie de choix de nœud est une voie de recherche prometteuse pour améliorer la performance des algorithmes de B&B sur intervalles. Nous avons proposé dans cet article deux nouvelles stratégies qui obtiennent de bons résultats expérimentaux sur des instances difficiles.

Ces deux approches sont basées sur un algorithme de recherche en meilleur d'abord. La première stratégie, appelée `LBvUB`, alterne entre la sélection d'un nœud avec la plus petite borne inférieure (LB) de l'objectif et d'un nœud avec la plus petite borne supérieure (UB). Le critère UB est biaisé pour favoriser les descendants du nœud où le dernier point réalisable a été trouvé. Les bornes inférieures et supérieures sont rendues plus précises par des opérations de contraction.

La seconde stratégie de choix de nœud, `Feasible-`

Gain	gain >5	gain [2,5]	gain [1.2,2]	gain [1.05,1.2]	équiv. [0.95,1.05]	perte [0.8,0.95]	perte [0.5,0.8]	perte <0.5
#instances	4	7	29	21	15	5	0	1

TABLE 3 – Détail des gains et pertes de `LBvUB` par rapport à LB. Les gains sont exprimés par $\frac{time(LB)}{time(LBvUB)}$. Les valeurs indiquent le nombre d'instances avec un gain dans un intervalle donné.

Gain	gain > 5	gain $[2, 5]$	gain $[1.2, 2]$	gain $[1.05, 1.2]$	équiv. $[0.95, 1.05]$	perte $[0.8, 0.95]$	perte $[0.5, 0.8]$	perte < 0.5
#instances	4	21	21	9	18	9	2	0

TABLE 4 – Détail des gains/pertes de FD par rapport à LB. Les gains sont exprimés par $\frac{time(LB)}{time(FD)}$. Les valeurs donnent le nombre d'instances ayant un gain dans un intervalle donné.

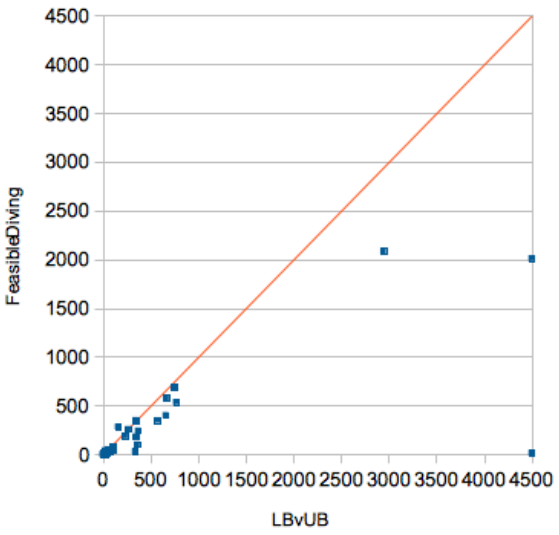


FIGURE 3 – `LBvUB` et `FD`. Les coordonnées de chaque point représentent les temps CPU (en secondes) utilisés par les compétiteurs.

`Diving`, est appelée à chaque nœud d'un algorithme B&B en meilleur d'abord en utilisant une variante du critère LB. La procédure `FeasibleDiving` plonge de manière gloutonne ne gardant qu'un nœud à chaque itération. Chaque nœud traité par cette procédure est contracté et un point réalisable y est cherché.

Ces deux stratégies donnent de meilleurs résultats que l'algorithme B&B standard en meilleur d'abord. `FeasibleDiving` permet de résoudre deux instances supplémentaires. Nous pensons que cette stratégie bénéficie d'une synergie avec les procédures de recherche de point réalisables disponibles dans `IbexOpt` et utilisées par la procédure `FeasibleDiving`.

Références

[1] I. Araya, G. Trombettoni, and B. Neveu. A Contractor Based on Convex Interval Taylor. In *Proc. CPAIOR*, volume 7298 of *LNCS*, pages 1–16. Springer, 2012.

[2] I. Araya, G. Trombettoni, B. Neveu, and G. Chabert. Upper Bounding in Inner Regions for Global Optimization under Inequality Constraints. *J. Global Optimization (JOGO)*, 60(2) :145–164, 2014.

[3] F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, volume 5649 of *LNCS*, pages 230–244. Springer, 1999.

[4] R.E. Bixby and E. Rothberg. Progress in Computational Mixed Integer Programming – A Look Back from the Other Side of the Tipping Point. *Annals of Operations Research*, 149 :37–41, 2007.

[5] P. Bonami, M. Kilink, and J. Linderoth. Algorithms and Software for Convex Mixed Integer Nonlinear Programs. Technical Report 1664, U. Wisconsin, 2009.

[6] L.G. Casado, J.A. Martinez, and I. Garcia. Experiments with a New Selection Criterion in a Fast Interval Optimization Algorithm. *Journal of Global Optimization*, 19 :247–264, 2001.

[7] G. Chabert. Interval-Based EXplorer, 2015. `www.ibex-lib.org`.

[8] G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173 :1079–1100, 2009.

[9] T. Csendes. New Subinterval Selection Criteria for Interval Global Optimization. *Journal of Global Optimization*, 19 :307–327, 2001.

[10] T. Csendes and D. Ratz. Subdivision Direction Selection in Interval Methods for Global Optimization. *SIAM Journal on Numerical Analysis*, 34(3), 1997.

Gain	gain > 5	gain $[2, 5]$	gain $[1.2, 2]$	gain $[1.05, 1.2]$	equiv. $[0.95, 1.05]$	perte $[0.8, 0.95]$	perte $[0.5, 0.8]$	perte < 0.5
#instances	2	10	24	12	25	5	5	1

TABLE 5 – Détail des gains/pertes de FD par rapport à `LBvUB`. Les gains sont exprimés par $\frac{time(LBvUB)}{time(FD)}$. Les valeurs sont les nombres d'instances ayant un gain dans un intervalle donné.

[11] A. Felner, S. Kraus, and R. E. Korf. KBFS : K-Best-First Search. *Annals of Mathematics and Artificial Intelligence*, 39, 2003.

[12] R.B. Kearfott and M. Novoa III. INTBIS, a Portable Interval Newton/Bisection Package. *ACM Trans. on Mathematical Software*, 16(2) :152–157, 1990.

[13] L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo Planning. In *Proc. ECML*, volume 4212 of *LNCS*, pages 282–293. Springer, 2006.

[14] M.C. Markot, J. Fernandez, L.G. Casado, and T. Csendes. New Interval Methods for Constrained Global Optimization. *Mathematical Programming*, 106 :287–318, 2006.

[15] F. Messine. *Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution des problèmes avec contraintes*. PhD thesis, LIMA-IRIT-ENSEEIHT-INPT, Toulouse, 1997.

[16] F. Messine and J.-L. Laganouelle. Enclosure Methods for Multivariate Differentiable Functions and Application to Global Optimization. *Journal of Universal Computer Science*, 4(6) :589–603, 1998.

[17] B. Neveu, G. Trombettoni, and I. Araya. Adaptive Constructive Interval Disjunction : Algorithms and Experiments. *Constraints Journal*, DOI : 10.1007/s10601-015-9180-3 :Accepted for publication, 2015.

[18] J. Ninin and F. Messine. A Metaheuristic Methodology Based on the Limitation of the Memory of Interval Branch and Bound Algorithms. *Journal of Global Optimization*, 50 :629–644, 2011.

[19] J. Ninin, F. Messine, and P. Hansen. A Reliable Affine Relaxation Method for Global Optimization. *4OR-Quaterly Journal of Operations Research*, 2014. Accepted for publication. DOI : 10.1007/s10288-014-0269-0.

[20] A. Sabharwal, H. Samulowitz, and C. Reddy. Guiding Combinatorial Optimization with UCT. In *Proc. CPAIOR*, volume 7298 of *LNCS*, pages 356–361. Springer, 2012.

[21] M. Tawarmalani and N. V. Sahinidis. Global Optimization of Mixed-Integer Nonlinear Programs : A Theoretical and Computational Study. *Mathematical Programming*, 99(3) :563–591, 2004.

[22] G. Trombettoni, I. Araya, B. Neveu, and G. Chabert. Inner Regions and Interval Linearizations for Global Optimization. In *Proc. AAAI*, pages 99–104, 2011.

[23] G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *Proc. CP*, volume 4741 of *LNCS*, pages 635–650. Springer, 2007.

A Liste des 84 instances d'optimisation globale sous contraintes testées

Nom	Br.	Name	Br.	Name	Br.	Nom	Br.
ex2_1_9	ssr	ex8_2_1	ssa	linear	ssr	hs088	lf
ex3_1_1	ssr	ex8_4_4	ssr	meanvar	ssr	hs093	ssr
ex5_3_2	ssr	ex8_4_5	lf	process	ssr	hs100	ssr
ex5_4_3	ssr	ex8_4_6	ssr	ramsey	lf	hs103	ssr
ex5_4_4	ssa	ex8_5_1	ssr	sambal	rr	hs104	lf
ex6_1_1	ssr	ex8_5_2	ssr	srcpm	sm	hs106	lf
ex6_1_3	ssr	ex8_5_6	ssr	avgasa	ssr	hs109	ssr
ex6_1_4	ssr	ex14_1_2	ssr	avgasb	ssr	hs113	lf
ex6_2_6	ssr	ex14_1_6	ssr	batch	ssa	hs114	rr
ex6_2_8	ssr	ex14_1_7	ssr	dipigri	ssr	hs117	ssa
ex6_2_9	ssr	ex14_2_1	ssr	disc2	ssr	hs119	ssa
ex6_2_10	ssr	ex14_2_3	ssr	dixchlng	lf	makela3	ssr
ex6_2_11	ssr	ex14_2_7	ssr	dualc1	ssr	matrix2	lf
ex6_2_12	ssr	alkyl (rr)	lf	dualc2	ssr	mistake	ssa
ex7_2_3	ssr	bearing	ssr	dualc5	ssr	odfits	ssr
ex7_2_4	lf	hhfair	ssr	genhs28	lf	optprloc	ssr
ex7_2_8	lf	himmel16	ssr	haifas	ssr	pentagon	ssr
ex7_2_9	lf	house	ssr	haldmads	lf	polak5	ssr
ex7_3_4	ssr	hydro	ssr	himmelbk	lf	robot	lf
ex7_3_5	ssr	immun	rr	hs056	lf	concon	ssr
ex8_1_8	ssr	launch	ssr	hs087	ssr	mconcon	ssr

TABLE 6 – Banc d'essai testé. Les systèmes ont été sélectionnés dans le banc d'essai Coconut (séries 1 et 2) suivant le protocole décrit à la partie 5. La meilleure stratégie de branchement (colonne Br.) a été choisie pour chaque système, la même pour toutes les méthodes, parmi : plus grand intervalle (lf), tour de rôle (rr), Smear max (sm), Smear sum absolue (ssa) (voir [12]) et Smear sum relative (ssr) (voir [22]).

Relations entre MDDs et Tuples et Modifications dynamique de contraintes MDDs

Guillaume Perez **Jean-Charles Régin**

Université Nice-Sophia Antipolis, CNRS, I3S UMR 7271, 06900 Sophia Antipolis, France

guillaume.perez06@gmail.com jcregin@gmail.com

Résumé

Nous étudions les relations entre Multi-Valued Decision Diagrams (MDD) et les tuples (i.e. les éléments du produit cartésien du domaine des variables). D'abord nous améliorons les méthodes existantes pour transformer un ensemble de tuples, des Global Cut Seeds et des séquences de tuples en MDDs. Puis nous présentons plusieurs algorithmes sur place pour ajouter et supprimer des tuples d'un MDD. Ensuite nous considérons une contrainte MDD qui est modifiée durant la recherche en supprimant plusieurs tuples. Nous donnons un algorithme qui adapte MDD-4R à ces modifications dynamiques et persistantes. Plusieurs expérimentations montrent que les contraintes MDD sont compétitives avec les contraintes de table.

Abstract

We study the relations between Multi-valued Decision Diagrams (MDD) and tuples (i.e. elements of the Cartesian Product of variables). First, we improve the existing methods for transforming a set of tuples, Global Cut Seeds, sequences of tuples into MDDs. Then, we present some in-place algorithms for adding and deleting tuples from an MDD. Next, we consider an MDD constraint which is modified during the search by deleting some tuples. We give an algorithm which adapts MDD-4R to these dynamic and persistent modifications. Some experiments show that MDD constraints are competitive with Table constraints.

1 Introduction

Les contraintes de table sont fondamentales et implémentées dans tous les solveurs CP. Elles sont explicitement définies par un ensemble d'éléments du produit cartésien du domaine des variables, aussi appelés tuples, qui sont valides.

Dans cet article, nous détaillons uniquement la phase de suppression d'un ensemble de tuples d'un MDD, pour le reste de ces méthodes, le lecteur est invité à lire [16].

Cheng et Yap ont proposé de compresser l'ensemble des tuples d'une contrainte en utilisant un Multi-valued Decision Diagram (MDD) et ont défini `mddc` un des premiers algorithme établissant la cohérence d'arc pour ceux-ci [7, 6]. Récemment, nous avons présenté MDD-4R un nouvel algorithme qui améliore `mddc` [14]. MDD-4R procède comme GAC-4R et, contrairement à `mddc`, maintient le MDD durant la recherche d'une solution. Nous avons aussi introduit un algorithme efficace pour réduire un MDD et plusieurs algorithmes pour combiner les MDDs [15]. Grâce à ces nouveaux algorithmes, plusieurs expérimentations basées sur des problèmes réels montrent qu'une approche basée sur les MDD devient compétitive avec les méthodes ad-hoc comme l'algorithme de filtrage associé avec les contraintes regular ou knapsack. De ce fait, le remplacement des contraintes de table par un MDD semble être possible dans le futur. Dans ce papier, nous espérons faire un pas de plus dans cette direction.

Les contraintes de table sont très utiles pour modéliser et résoudre beaucoup de problèmes réels. Elles peuvent être spécifiées soit directement par l'utilisateur, soit en synthétisant d'autres contraintes ou sous-problèmes [13, 12]. Elles ont été modifiées pour travailler avec des tuples ou des séquences de tuples [8, 10, 17]. De plus, leur expressivité est forte.

Si nous voulons être compétitif avec les contraintes de table, nous avons besoin de représenter efficacement différents type d'ensembles de tuples compressés. Pour cela nous montrons comment les GCSs (Global Cut Seeds) et les séquences de tuples peuvent être représentés par un MDD. Notamment, nous allons montrer qu'un MDD peut être construit directement depuis un ensemble trié de tuples sans utiliser de structure intermédiaire coûteuse en espace

comme celle proposée par Cheng and Yap.

Ensuite, nous considérons l'ajout et la suppression d'un tuple d'un MDD. Nous verrons que cette opération peut être efficacement effectuée en utilisant une méthode qui consiste à isoler le chemin du MDD correspondant au tuple dans le cas de la suppression et a celui du préfixe commun dans le cas de l'ajout. Ces opérations rendent plus facile l'ajout/suppression d'un ensemble de tuples, qui sont les modifications que nous pouvons apporter à un MDD. D'un coté, cela renforce l'expressivité des MDDs. D'un autre coté, cela ouvre aussi la porte à des algorithmes dynamiques pour maintenir la cohérence d'arc des contraintes MDD. Aussi nous proposons un algorithme de cohérence d'arc pour ces contraintes quand plusieurs tuples sont définitivement supprimés durant la recherche. Cet algorithme est basé sur les opérations effectuées au préalable et doit être implémenté efficacement car la suppression d'un tuple peut créer de nouveaux noeuds dans le MDD et cela cause plusieurs problème avec la restaurations après le backtrack. En d'autres termes, une suppression persistante est une modification monotone par rapport à la consistance de la contrainte, mais la maintenance du MDD n'est pas monotone.

Être capable de maintenir la cohérence d'arc d'un MDD auquel plusieurs tuples sont supprimés définitivement possède deux gros avantages. D'abord, cela est utile pour travailler avec des problèmes comme l'enregistrement des nogoods. Actuellement, des algorithmes ad-hoc ou des tables dynamiques sont utilisés. Donc cela renforce notre idée de voir les contraintes MDD comme un possible remplacement des contraintes de table. Ensuite, les contraintes MDD sont maintenant compétitives avec les algorithmes ad-hoc pour la contrainte regular. Donc avoir un algorithme dynamique pour eux nous donne immédiatement un algorithme dynamique pour les contraintes regular.

Dans cet article, nous rappelons quelques définitions, nous détaillons uniquement la suppression sur place de tuples dans un MDD. Nous validons notre approche à l'aide d'une étude expérimentale et pour finir concluons. Les opérations d'ajout de tuple sur place sont détaillé dans le rapport [16].

2 Définitions

Un multi-valued decision diagram (MDD) est une méthode pour représenter une fonction discrète. C'est une extension multi-valeur des BDDs [5].

Un MDD, comme utilisé en CP, [1, 11, 12, 3, 9], est un Directed Acyclic Graph (DAG) avec une racine, utilisé pour représenter des fonctions $f : \{0...d-1\}^r \rightarrow \{true, false\}$, basées sur un entier donné d (Voir Figure .). Pour r variables données, la représentation par un DAG est faite pour contenir r niveaux, tel que chaque variable soit représentée par un niveau spécifique du graphe. Chaque

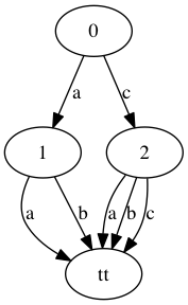


FIGURE 1 – Un MDD representant l'ensemble de tuples {{a,a},{a,b},{c,a},{c,b},{c,c}}

noeud d'un niveau spécifique possède au plus d arcs sortants vers les noeuds du niveau suivant du graphe. Chaque arc possède une étiquette correspondant à sa valeur. Le niveau final est représenté par le noeud terminal true (le noeud terminal false est souvent omis). Il y a une équivalence entre $f(v_1, ..., v_r) = true$ et l'existence d'un chemin du noeud racine au noeud terminal true et dont les arcs sont étiquetés par $v_1, ..., v_r$. Les noeuds sans aucun arc sortant ou arc entrant sont supprimés.

Dans une contrainte MDD, le MDD modélise l'ensemble des tuples qui satisfont la contrainte, tel que chaque chemin du noeud racine au noeud terminal true correspond à un tuple autorisé. Chaque variable du MDD correspond à une variable de la contrainte. Un arc associé à une variable du MDD correspond à une valeur de la variable correspondante de la contrainte.

Par commodité, nous allons noter d le nombre maximum de valeur dans le domaine d'une variable et un arc sortant de x vers y étiqueté par v sera noté (x, v, y). Nous noterons aussi par $\omega^+(n)$ l'ensemble des arcs sortants de n.

Un exemple de MDD est donné en Figure 1. Ce MDD représente les tuples {a,a}, {a,b}, {c,a}, {c,b} et {c,c}. Pour chaque tuple, il y a un chemin du noeud racine (0) vers le noeud terminal (tt) dont les arcs sont étiquetés par les valeurs du tuple.

La réduction d'un MDD est une des opérations les plus importante. Elle consiste à fusionner les noeuds équivalents, i.e. les noeuds possédant le même ensemble ω^+ en ne considérant que les deux dernières composantes (valeur et extrémité terminale). Généralement, un algorithme de réduction fusionne les noeuds jusqu'à ce qu'il n'y ait plus de noeuds équivalents. Le plus souvent, seuls les MDD réduits sont considérés car ils sont plus petits. Notons que l'opération de réduction ne peut pas augmenter le nombre d'arc.

3 Suppression de tuples d'un MDD

Plusieurs travaux ont été effectués pour appliquer des opérations sur les BDDs. Par exemple, Bryant a définit des algorithmes pour appliquer différents opérateurs [5, 4]. Ce-

pendant, les algorithmes décrits ne sont pas sur place (i.e. il y a la creation d'un BDD résultat) et ce n'est pas facile de généraliser ces algorithmes écrit pour des BDDs aux MDDs. Cela est en partie due aux règles booléennes qui ne sont plus vraies quand nous avons d valeurs dans le domaine et parce-que la complexité de certains d'entre eux est multiplié par $O(d)$ quand nous travaillons avec d valeurs. Plusieurs algorithmes ont été proposés pour appliquer des opérateurs aux MDDs [2, 15]. Cependant, ils ne sont pas sur place.

Dans cette section nous définissons un algorithme sur place pour la suppression de tuples d'un MDD. Ces algorithmes sont nécessaires pour être capable de définir des algorithmes dynamiques pour la contrainte MDD compétitifs avec ceux des contraintes de table.

Nous donnons d'abord un algorithme pour la suppression d'un seul tuple d'un MDD. Puis nous le généralisons pour un ensemble de tuples.

3.1 Suppression d'un tuple d'un MDD

La suppression d'un tuple d'un MDD est faite par une opération appelée "path isolation". L'idée de cette opération est de construire un chemin spécifique dont les arcs sont étiquetés par les valeurs du tuple qui doit être supprimé. De plus les arcs équivalents aux arcs du chemin isolé sont supprimés du MDD.

Cela se déroule en quatre étapes :

1. L'isolation du premier niveau
2. L'isolation de tous les niveaux intermédiaires (ni le premier, ni le dernier)
3. L'isolation du dernier niveau
4. L'appel d'un algorithme de réduction incrémentale sur le MDD

Détaillons ces étapes. Soit τ le tuple que nous voulons supprimer. soit $\tau[i]$ la valeur pour la variable $x[i]$.

Étape 1. D'abord nous identifions $a_1 = (r, n_1, \tau[1])$ l'arc du premier niveau étiqueté par $\tau[1]$ la première valeur du tuple. Nous créons le noeud ne_1, l'arc $(r, ne_1, \tau[1])$ et nous supprimons l'arc a_1. Nous affectons $mddNode$ à n_1 et $isolatedNode$ à ne_1.

Étape 2. Pour chaque niveau i de 2 à $r-1$ nous répétons l'opération suivante. Nous identifions $a_i = (mddNode, n_{i+1}, \tau[i])$ l'arc sortant partant de $mddNode$ étiqueté par $\tau[i]$. Nous créons le noeud ne_{i+1} et l'arc $(isolatedNode, ne_{i+1}, \tau[i])$. Pour chaque arc $(mddNode, y, w)$ tel que $w \neq \tau[i]$ nous créons l'arc $(isolatedNode, y, w)$. Nous affectons $mddNode$ à n_{i+1} et $isolatedNode$ à ne_{i+1}.

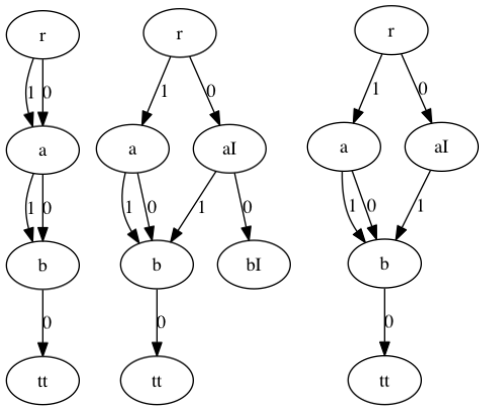


FIGURE 2 – le MDD de gauche est le MDD initial auquel nous retirons le tuple {0,0,0}. Le MDD du milieu MDD est le résultat de l'isolation du chemin correspondant au tuple. les noeuds aI et bI sont crées depuis les noeuds a et b dans le but de créer le chemin isolé. Le MDD de droite est le résultat de cette opération.

Étape 3. Pour chaque arc $(mddNode, tt, w)$ tel que $w \neq \tau[i]$ nous créons l'arc $(isolatedNode, tt, w)$.

Étape 4. Nous appliquons un algorithme de réduction en ne considérant que le chemin et les voisins du chemin.

Si à un moment nous ne pouvons plus identifier un arc, cela signifie que τ n'appartient pas au MDD. Figure 2 montre l'application de cet algorithme.

la complexité de cette suppression est bornée par $O(rd)$ car pour chaque noeud isolé, nous avons besoin de recréer ses arcs. cependant, en pratique c'est souvent proche de $O(r)$. Donc nous pouvons simplement gérer la suppression d'un ensemble de tuples en répétant cet algorithme. Nous proposons d'améliorer cette méthode.

3.1.1 Suppression d'un ensemble de tuples.

Nous donnons un algorithme sur place pour supprimer un ensemble de tuples d'un MDD. Dans ce cas, nous transformons l'ensemble de tuples en MDD puis nous soustrayons ce nouveau MDD du MDD initial. Cet algorithme généralise le précédant et suit les étapes suivantes. Il isole les noeuds ayants un chemin commun au deux MDD, ensuite il supprime les arcs communs des noeuds isolés au dernier niveau du second. L'algorithme 1 est une possible implémentation.

La Figure 3 montre la soustraction de l'ensemble de tuples { {1,0,1} ,{1,1,1} ,{1,2,1} ,{1,3,1} } au MDD représentant tout les tuples possibles pour les valeurs {0,1,2,3}. L'ensemble est isolé du MDD. Ensuite, la suppression de

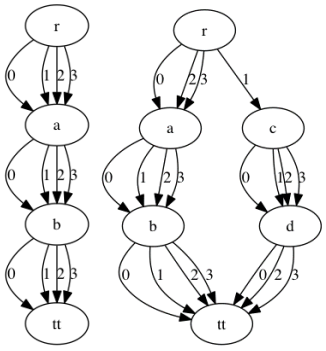


FIGURE 3 – Le MDD de gauche représente le MDD autorisant tous les tuples pour le domaine {0,1,2,3}. Le MDD de droite représente la suppression de l'ensemble de tuples { {1,0,1} ,{1,1,1} ,{1,2,1} ,{1,3,1} } au MDD de gauche.

l'arc étiqueté 1 du noeud d correspond à la suppression seule des tuples contenus dans le l'ensemble.

Il est difficile de borner la complexité de la suppression de T tuples, car le MDD créé peut compresser l'information.

3.2 Réduction incrémentale

Une réduction est requise après la suppression ou l'ajout de tuples. En utilisant un algorithme générique, cela est coûteux car tous les noeuds sont traversés pour pouvoir fusionner les noeuds équivalents. Si l'on considère que nous ajoutons/supprimons des tuples d'un MDD qui est déjà réduit, nous pouvons économiser des calculs pour l'application de la réduction appliquée après ces opérations. En effet seuls les noeuds isolés et ceux ayant un noeud voisin isolé ont besoin d'être reconsidéré pour tester leur équivalence car les autres ne sont pas modifiés. De plus, identifier les noeuds isolés est facile car ils appartiennent à la liste L de l'algorithme. L'avantage de cette approche est que l'étape de réduction n'augmente pas la complexité de l'opération de suppression.

4 Experimentations

Le but de ces experimentations est de montrer que lorsque nous appliquons des suppressions une approche basée sur les MDD est compétitive avec une approche basée sur les tables. Même si elle utilise des tuples compressés.

Machines MacBook Pro, Intel Core I7, 2,3GHz, 8GB memory.

Solveur or-tools 3158.

Algorithm 1: In-place Deletion Algorithm

```
DELETION(L, mdd_1, mdd_2)
  // Step 1 : first layer
  for each (root(mdd_1), v, y_1) ∈ ω⁺(root(mdd_1)) do
      if ∃(root(mdd_2), v, y_2) ∈ ω⁺(root(mdd_2)) then
          ADDARCANDNODE(L, 1, root(mdd_1), v, y_1, y_2)
          DELETEARC(root(mdd_1), v, y_1)
  // Step 2 : intermediate layers
  // L[i] is the set of nodes in layer i.
  for each i ∈ 1..r − 2 do
      L[i] ← ∅
      for each node x ∈ L[i − 1] do
          get x_1 and x_2 from x = (x_1, x_2)
          for each (x_1, v, y_1) ∈ ω⁺(x_1) do
              if ∃(x_2, v, y_2) ∈ ω⁺(x_2) then
                  ADDARCANDNODE(L, i, x, v, y_1, y_2)
              else CREATEARC(L, i, x, v, y_1)
  // Step 3 : last layer
  for each node x ∈ L[r − 1] do
      get x_1 and x_2 from x = (x_1, x_2)
      for each (x_1, v, tt) ∈ ω⁺(x_1) do
          if / ∃(x_2, v, y_2) ∈ ω⁺(x_2) then
              CREATEARC(L, r, x, v, tt)
  PREDUCE(L)
  return root

ADDARCANDNODE(L, i, x, y_1, v, y_2)
  if ∄y ∈ L[i] s.t. y = (y_1, y_2) then
      y ← CREATENODE(y_1, y_2)
      add y to L[i]
  CREATEARC(x, v, y)
```

Instances selectionnées Nous construisons des instances aléatoires pour avoir une vue globale des deux approches.

Impact des suppressions Nous étudions le nombre de modifications déclenchées par la suppression de tuples. L'ensemble de tuples contient 6 variables et 5 valeurs. La Figure 4 montre les résultats pour trois types d'ensemble de tuples, un ayant une petite densité (8%), une moyenne densité (15%) et une grande densité (25%). Nous observons une évolution linéaire pour les contraintes de table ce qui est normal. L'approche MDD a besoin de moins de modifications. Étonnement, les meilleurs résultats sont obtenus pour les faibles et hautes densités. Le pire des cas pour les MDDs semble pour une densité moyenne. Cela peut être expliqué par le fait que les modifications ont beaucoup à faire et qu'il y a beaucoup de noeuds. La haute densité est fortement compressée, alors que la faible densité n'a que

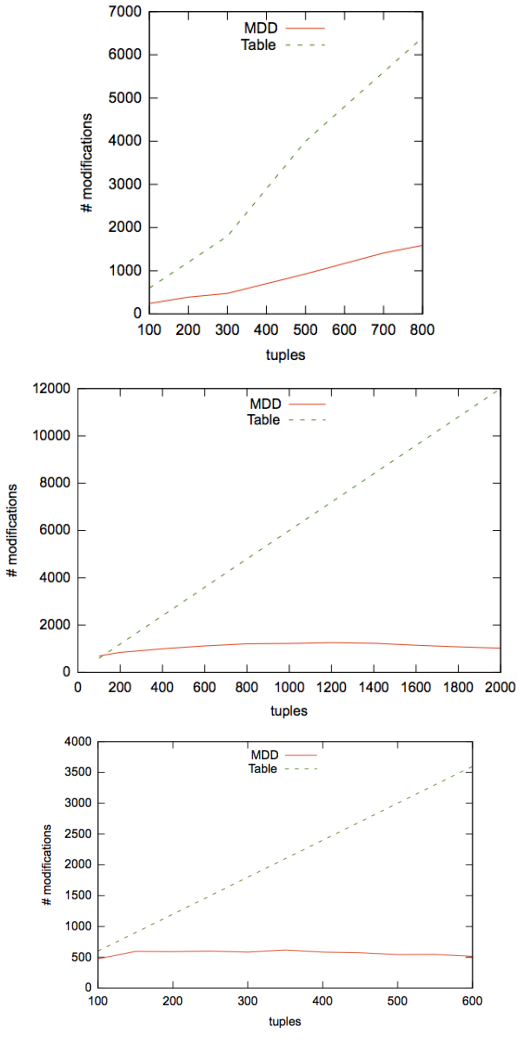


FIGURE 4 – Nombre de modifications pour la suppression de tuples. Graphe du haut : medium density. Graphe du bas : high density. Graphe du milieu : high density

Algorithme	#suppression	Domain size		
		10	12	14
GAC4R	0	1570	2123	2710
GAC4R	100 000	1274	1511	1763
MDD4RD	0	1115	1385	1815
MDD4RD	100 000	763	879	1064

TABLE 1 – temps (en ms) pour supprimer 100 000 tuples depuis un ensemble de tuples contenant 230 000 éléments durant la recherche de toutes les solutions.

peu de travail à effectuer.

Suppression de tuples durant la recherche Nous considérons un problème composé de contraintes d'arité 6. Chaque contrainte est définie depuis une table contenant 230 000 tuples. Nous cherchons toutes les solutions et appliquons la suppression de 100 000 tuples durant la recherche. Cela déclenche 600 000 modifications pour les contraintes de table, car nous avons besoin de vérifier si un tuple est un support pour chacune de ses valeurs. Il est intéressant de remarquer que le nombre de modifications (création/suppression d'arcs et de noeuds) déclenché pour le MDD est de 135 000. Ce nombre est donc plus petit, cela vient du fait que un MDD compresse les tuples. Par contre, plus d'opérations peuvent être requises quand un tuple est supprimé mais la structure de donnée reste compressés et reste donc plus performante. La Table 1 nous donne plusieurs informations sur le temps requis pour appliquer ces opérations. Une fois encore, l'approche MDD fonctionne bien.

5 Conclusion

Nous avons décrit des algorithmes sur place efficaces pour supprimer un ensemble de tuples d'un MDD. Nous avons aussi montré plusieurs experimentations. Ce travail contribue à la preuve que l'approche basée sur les MDD est compétitive avec une approche basée sur les tables pour représenter les contraintes en extension en programmation par contraintes.

Références

[1] Henrik Reif Andersen, Tarik Hadzic, John N. Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *CP*, pages 118–132, 2007.

[2] D. Bergman, A. Cire, and W-J. van Hoeve. Mdd propagation for sequence constraints. *Journal of Artificial Intelligence Research*, 50 :697–722, 2014.

[3] David Bergman, Willem Jan van Hoeve, and John N. Hooker. Manipulating mdd relaxations for combinatorial optimization. In *CPAIOR*, pages 20–35, 2011.

[4] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3) :293–318, 1992.

[5] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C35(8) :677–691, 1986.

[6] K. Cheng and R. Yap. An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15, 2010.

[7] Kenil C. K. Cheng and Roland H. C. Yap. Maintaining generalized arc consistency on ad hoc r-ary constraints. In *CP*, pages 509–523, 2008.

[8] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proc. CP'01*, pages 77–92, Paphos, Cyprus, 2001.

[9] G. Gange, P. Stuckey, and Radoslaw Szymanek. Mdd propagators with explanation. *Constraints*, 16 :407–429, 2011.

[10] I. Gent, C. Jefferson, I. Miguel, and P. Nightingale. Data structures for generalised arc consistency for extensional constraints. In *Proc. AAAI'07*, pages 191–197, Vancouver, Canada, 2007.

[11] Tarik Hadzic, John N. Hooker, Barry O'Sullivan, and Peter Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In *CP*, pages 448–462, 2008.

[12] Samid Hoda, Willem Jan van Hoeve, and John N. Hooker. A systematic approach to mdd-based constraint programming. In *CP*, pages 266–280, 2010.

[13] Olivier Lhomme. Practical reformulations with table constraints. In *ECAI*, pages 911–912, 2012.

[14] G. Perez and J-C. Régin. Improving GAC-4 for table and MDD constraints. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 606–621, 2014.

[15] G. Perez and J-C. Régin. Efficient operations on mdds for building constraint programming models. In *International Joint Conference on Artificial Intelligence, IJCAI-15*, Argentina, 2015.

[16] G. Perez and J-C. Régin. Relations between mdds and tuples and dynamic modifications of mdds based constraints. *CoRR*, abs/1505.02552, 2015.

[17] J-C. Régin. Improving the expressiveness of table constraints. In *CP'11, proceedings workshop ModRef'11*, 2011.

Un langage orienté parallèle pour modéliser des solveurs de contraintes

Alejandro REYES AMARO Eric MONFROY Florian RICHOUX

LINA - UMR 6241, TASC - INRIA Université de Nantes, France.
{alejandro.reyes, eric.monfroy, florian.richoux}@univ-nantes.fr

Résumé

Cet article présente Parallel-Oriented Solver Language (POSL, prononcé "*puzzle*") : un système pour construire des méta-heuristiques interconnectées travaillant en parallèle. Le but de ce travail est d'obtenir un système pour facilement construire des solveurs et réduire l'effort de leur développement en proposant un mécanisme de réutilisation de code entre les différents solveurs. La nouveauté de cette approche porte sur le fait que l'on voit un solveur comme un ensemble de composants spécifiques, écris dans un langage orienté parallèle basé sur des opérateurs.

Un avantage de POSL est la possibilité de partager non seulement des informations mais aussi des comportements, permettant ainsi de modifier à chaud les solveurs. POSL permets aux composants d'un solveur d'être transmis et exécutés par d'autres solveurs. Il propose également une couche supplémentaire permettant de définir dynamiquement des connexions entre solveurs.

L'implémentation de POSL restant un travail en cours, cet article se concentre uniquement sur ses concepts.

1 Introduction

L'optimisation combinatoire a d'importantes applications dans des champs divers, tels que l'apprentissage automatique, l'intelligence artificielle et l'ingénierie logicielle. Dans certains cas, le but principal est de simplement trouver une solution, comme pour les *problèmes de satisfaction de contraintes (CSP)*. Une solution sera une affectation de valeur à chaque variable du problème afin de satisfaire chacune de ses contraintes. En d'autres mots : il s'agit de chercher une solution possible.

Les *CSP* trouvent beaucoup d'applications dans l'industrie. Pour cette raison, plusieurs techniques et méthodes peuvent être appliquées pour résoudre ces problèmes. Bien que ces techniques, telles que les méta-heuristiques par exemple, se montrent efficaces, les problèmes que l'on souhaite résoudre en pratique sont parfois trop vastes, c'est-à-dire, avec un espace de recherche trop grand, pour pouvoir être traité en un temps raisonnable.

Cependant, le développement de l'architecture des ordinateurs nous mène vers des machines massivement *multi/many cœurs*. Cette évolution, étroitement liée aux développements des super-calculateurs, a ouvert une nouvelle manière de trouver des solutions pour les problèmes d'optimisation combinatoire, réduisant les temps de résolution. Adaptive Search [3] est l'un des algorithmes les plus efficaces, montrant de très bonnes performances et passant à l'échelle de plusieurs centaines ou même milliers de cœurs. C'est un exemple de recherche locale dit *multi-walk*, c'est-à-dire d'algorithmes explorant l'espace de recherche en lançant plusieurs processus indépendants de recherche, avec ou sans communication. Pour Adaptive Search, une implémentation de multi-walk coopératif a été publiée dans [9]. Ces travaux ont montré l'efficacité du schéma parallèle multi-walk, c'est pourquoi nous avons orienté POSL vers celui-ci.

Ces dernières années, beaucoup d'efforts ont été fait en parallélisation de la programmation par contraintes. Dans ce champ de recherche, la bonne gestion des communications inter-processus, utiles pour partager des informations entre solveurs, est absolument critique pour garantir de bonnes performances. Dans [8], une idée proposée est d'inclure des composants bas niveau de raisonnement pour la résolution de problèmes SAT, afin d'ajuster dynamiquement la taille des clauses partagées pour réduire les potentiels engorgements de communication. L'interaction entre solveurs est appelée *coopération de solveurs*

et est très populaire dans ce domaine de recherche du fait de leurs bons résultats [11]. *Meta–S* est une implémentation du système proposé dans [5], permettant de traiter des problèmes à travers la coopération de solveurs sur des domaines à spécialiser. POSL propose un mécanisme de création de stratégies de communication indépendamment des solveurs, donnant la possibilité d'étudier facilement les processus de résolution et leurs résultats. Créer des solveurs implémentant différentes stratégies parallèles peut être aussi complexe que fastidieux. En ce sens, POSL donne la possibilité de faire de rapides prototypes de solveurs parallèles.

En programmation par contraintes, les résultats publiés se focalisent souvent sur l'amélioration de certaines métriques telles la vitesse de convergence ou la qualité des solutions par des solveurs connus. Cependant, ceci requière de profondes connaissances pour trouver le bon algorithme à appliquer au bon problème. HYPERION [2] est un système Java pour méta et hyper-heuristiques basé sur le principe d'interopérabilité, fournissant des patrons génériques pour une variété d'algorithmes de recherche locale et évolutionnaires, permettant des prototypages rapides avec la possibilité de réutiliser le code source. POSL vise à offrir ces avantages, mais en fournissant également un mécanisme permettant de définir des protocoles de communications entre solveurs.

Dans cet article, nous expliquons une méthodologie pour utiliser POSL afin de construire différents solveurs coopératifs basés sur le couplage de quatre composants indépendants : *operation module*, *open channel*, *computation strategy* et les *communication channels* ou *subscription*. Récemment, l'approche hybride mène à de très bons résultats en satisfaction de contraintes [4]. Ainsi, puisque les composants de solveurs peuvent être combinés, POSL est conçu pour obtenir simplement des ensembles de solveurs différents à exécuter en parallèle.

POSL propose, à travers un simple langage basé sur des opérateurs, une manière de créer une *computation strategy*, combinant des *composants* (à savoir des *operation modules* et *open channels*) définis au préalable. On retrouve une idée similaire dans [6] sans communication, où une méthode d'algorithme évolutionnaire utilisant un simple opérateur de composition est proposée, afin d'instancier automatiquement de nouvelles heuristiques de recherche locale pour SAT, en voyant ces dernières comme une combinaison de blocs indépendants. Une autre idée intéressante est proposée dans TEMPLAR [12], un système pour générer des algorithmes changeant de composants prédéfinis via une hyper-heuristique Dans la dernière phase de programmation via POSL, les solveurs peuvent être connectés à d'autres solveurs, en fonction de la structure de leurs *open channels*, leur permettant de partager non seulement des informations mais également des comportements, donnant la possibilité d'envoyer et recevoir leur *operation module*. Cette approche donne ainsi aux solveurs la possibilité d'évoluer durant leur exécution, en fonction d'un contexte parallèle.

Les travaux présentés dans ce papier se concentrent sur les concepts de POSL uniquement ; son implémentation est un travail en cours.

2 Problèmes ciblés

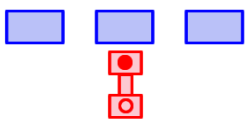
POSL est un système pour résoudre les CSP. Un CSP est défini par un triplet $\langle X, D, C \rangle$ où $X = \{x_1, x_2, \ldots, x_n\}$ est un ensemble fini de variables, $D = \{D_1, D_2, \ldots, D_n\}$, est l'ensemble des domaines de chaque variable dans X, et $C = \{c_1, c_2, \ldots, c_m\}$, est l'ensemble des contraintes. Chaque contrainte est définie selon un ensemble de variables et détermine les combinaisons possibles de leurs valeurs. Nous désignons par $\mathcal{P}$ un *CSP* donné.

Une configuration $s \in D_1 \times D_2 \times \cdots \times D_n$ est une combinaison de valeurs de chaque variable dans X. Nous écrivons que s est une solution de $\mathcal{P}$ si et seulement si s satisfait chacune des contraintes $c_i \in C$.

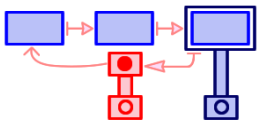
3 Solveurs parallèles POSL

POSL permet de construire des solveurs suivant différentes étapes :

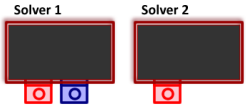
1. L'algorithme du solveur considéré est exprimé via une décomposition en modules de calcul. Ces modules sont implémentés à la manière de *fonctions* séparées. Nous appelons *operation module* ces morceaux de calcul (figure 1a).
2. L'étape suivante consiste à décider quelles sont les types d'informations que l'on souhaite recevoir des autres solveurs. Ces informations sont encapsulées dans des composants appelés *open channel*, permettant de transmettre des données entre solveurs (figure 1a).
3. Une *stratégie générique* est codée à travers POSL, en utilisant les opérateurs fournis par le langage appliqués sur les composants donnés lors des étapes 1 et 2. Cette stratégie définie non seulement les informations échangées, mais détermine également l'exécution parallèle de composants. Lors de cette étape, les informations à partager sont transmises via les opérateurs ad-hoc. On peut voir cette étape comme la définition de la colonne vertébrale des solveurs.



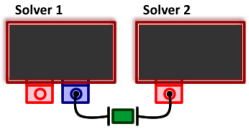
(a) Definition du *operation module* s (blocs bleus) et l'*open channel* s (rouge)



(b) Definition de la *computation strategy*



(c) Solveurs déconnectés



(d) Definition de cooperation par *subscription*s

FIGURE 1 – Construire des solveurs parallèles avec POSL

4. Les solveurs sont créés en instanciant la strategy, *operation module* et *open channel*, puis en les assemblant (figure 1d).

Les sous-sections suivantes expliquent en détail chacune des étapes ci-dessus.

3.1 Operation module

Un *operation module* est la plus basique et abstraite manière de définir un composant de calcul. Il reçoit une entrée, exécute un algorithme interne et retourne une sortie. Dans ce papier, nous utilisons ce concept afin de décrire et définir les composants de base d'un solveur, qui seront assemblés par la *computation strategy*.

Un *operation module* représente un morceau de l'algorithme du solveur qui est suceptible de changer au cours de l'exécution. Il peut être dynamiquement remplacé ou combiné avec d'autres *operation modules*, puisque les *operation modules* sont également des informations échangeables entre les solveurs. De cette manière, le solveur peut changer/adapter son comportement à chaud, en combinant ses *operation modules* avec ceux des autres solveurs. Ils sont représentés par des blocs bleus dans la figure 1.

Definition 1 ***(Operation Module)*** *Un operation module Om est une application définie par :*

$$Om : \mathcal{D} \rightarrow \mathcal{I} \tag{1}$$

Dans (1), la nature de $\mathcal{D}$ et $\mathcal{I}$ dépend du type d'*operation module*. Ils peuvent être soit une configuration, ou un ensemble de configurations, ou un ensemble de valeurs de différents types de données, etc.

Soit une méta-heuristique de recherche locale, basée sur un algorithme bien connu, comme par exemple *Tabu Search*. Prenons l'exemple d'un *operation module* retournant le voisinage d'une configuration donnée, pour une certaine métrique de voisinage. Cet *operation module* peut être défini par la fonction suivante :

$$Om_{neighborhood} : D_1 \times D_2 \times \cdots \times D_n \rightarrow 2^{D_1 \times D_2 \times \cdots \times D_n}$$

où D_i représente la définition des domaines de chacune des variables de la configuration d'entrée.

3.2 Open channels

Les *open channel* sont les composants des solveurs en charge de la réception des informations communiquées entre solveurs. Ils peuvent interagir avec les *operation modules*, en fonction du *computation strategy*. Les *open channel* jouent le rôlé de prise, permettant aux solveurs de se brancher et de recevoir des informations. Il sont représentés en rouge dans la figure 8.

Un *open channel* peut recevoir deux types d'informations, provenant toujours d'un solveur tiers : des données et des *operation modules*. En ce qui concerne les *operation modules*, leur communication peut se faire via la transmission d'identifiants permettant à chaque solveur de les instancier.

Pour faire la distinction entre les deux différents types de *open channels*, nous appelons **Data Open Channels** les *open channels* responsables de la réception de données et **Object Open Channels** ceux s'occupant de la réception et de l'instanciation d'*operation modules*.

Definition 2 ***(Data Open Channel)*** *Un* Data Open Channel *Ch est un composant produisant une application définie comme suit :*

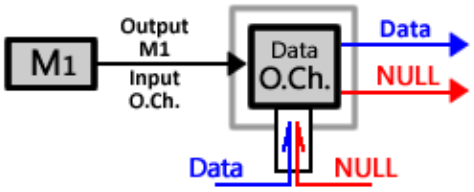
$$Ch : \mathcal{U} \rightarrow \mathcal{I} \tag{2}$$

et retournant l'information $\mathcal{I}$ provenant d'un sovleur tiers, quelque soit l'entrée $\mathcal{U}$.

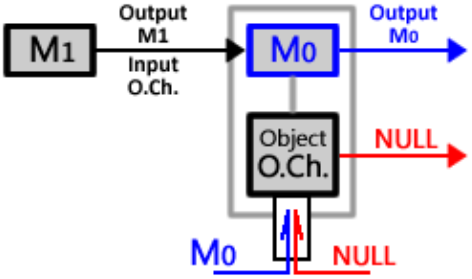
Definition 3 ***(Object Open Channel)*** *Si nous notons $\mathbb{M}$ l'espace de tous les operation modules de la définition 1, alors un* Object Open Channel *Ch est un composant produisant un operation module venant d'un solveur tiers défini ainsi :*

$$Ch : \mathbb{M} \rightarrow \mathbb{M} \tag{3}$$

Puisque les *open channel* reçoivent des informations provenant d'autres solveurs sans pour autant avoir de contrôle sur celles-ci, il est nécessaire de définir l'information *NULL*, signifiant l'absence d'information. La figure 2 montre le mécanisme interne d'un *open channel*. Si un **Data Open Channel** reçoit une information, celle-ci est automatiquement retournée



(a) Data Open Channel



(b) Object Open Channel

FIGURE 2 – Mécanisme interne du *open channel*

(figure 2a, lignes bleues). Si un **Object Open Channel** reçoit un *operation module*, ce dernier est instancié et exécuté avec l'entrée de l'*open channel*, et le résultat est retourné (figure 2b, lignes bleues). Dans les deux cas, si aucune information n'est reçue, l'*open channel* retourne l'objet *NULL* (figure 2, lignes rouges).

3.3 Computation strategy

La *computation strategy* est le cœur du solver. Elle joint les *operation modules* et les *open channels* de manière cohérente, tout en leur restant indépendant. Ceci signifie qu'elle peut changer ou être modifiée durant l'exécution, sans altérer l'algorithme général et en respectant la structure du solveur. À travers la *computation strategy*, on peut décider également des informations à envoyer aux autres solveurs.

La *computation strategy* est définie par des opérations paramétriques. Tout d'abord, nous allons définir ce que sont ces opérations et les illustrées dans différents scénarios.

Definition 4 ***(Opérateurs de*** POSL***)*** *Nous appelons* opérateur de POSL *un opérateur paramétrique défini comme suit :*

$$OP\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\} : \mathcal{D}_{OP} \rightarrow \mathcal{I}_{OP} \qquad (4)$$

Nous appelons Compound Module *l'opération définie par (4), où chaque $\mathcal{M}_i$ est : i) Soit un operation module, ii) Soit un open channel, iii) Soit un compound module.*

Par soucis de simplification, nous utiliserons à partir de maintenant le terme de Module *pour désigner soit un operation module, un open channel ou un compound module.*

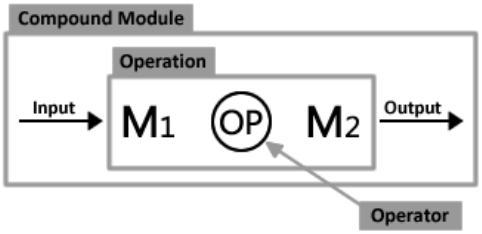


FIGURE 3 – Un *compound module*

Pour illustrer la définition 4, la figure 3 montre graphiquement les concepts d'*Operator*, *Operation* et *Compound Module*.

Chaque fois qu'une opération (4) est exécutée, l'opérateur *évalue* chaque *module* séparément, en affectant ses entrées à des *module* composant cette opération, c'est-à-dire que l'entrée de l'opérateur sera l'entrée de certains *modules* impliqués, voir de tous les *module* en fonction de l'opérateur. Après avoir évalué les *modules*, l'opérateur traite les données et retourne une sortie.

Le *processus d'évaluation* d'un *module* fonctionne de la manière suivante :

1. **Évaluation d'un Compound Module :** L'entrée de l'opérateur est passée au *compound module* et est évaluée.
2. **Évaluation d'un Operation Module :** Tout comme pour le *compound module*, l'entrée de l'operateur est passée à l'*operation module* et est évaluée.
3. **Évaluation d'un Open Channel :**
   — Data *open channel* : Le résultat de l'évaluation est la donnée réceptionnée, quelque soit l'entrée.
   — Object *open channel* : Le résultat de l'évaluation correspond à celui de l'évaluation de l'*operation module* réceptionné.

Comme nous le voyons par la définition 4, dans chaque opérateur fourni dans POSL, un ou plusieurs *compound modules* sont impliqués. En fonction de la nature de l'opérateur, ils peuvent être exécutés de manière séquentielle uniquement ou peuvent être exécutés en parallèle. POSL propose également des groupages d'opérations et laisse la possibilité de définir comment les *compound modules* impliqués vont être exécutés :

1. $[\mathrm{OP}\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}]$: Les *compound modules* $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n$ seront exécutés séquentiellement.
2. $[\![\mathrm{OP}\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}]\!]_p$: Les *compound module* $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n$ seront exécutés en parallèle si et seulement si *OP* supporte le parallélisme.

Dans le cas particulier où un des *compound modules* impliqués est un *open channel*, chaque opérateur gère l'information *NULL* à sa manière.

Afin de grouper des modules, nous utiliserons la notation $|.|$ comme un groupe générique qui pourra être indifféremment interprété comme $[.]$ ou comme $[\![.]\!]_p$.

L'opérateur suivant nous permet d'exécuter deux modules séquentiellement, l'un après l'autre.

Definition 5 ***(Operator Sequential Execution)*** *Soient i)* $\mathcal{M}_1 : \mathcal{D}_1 \rightarrow \mathcal{I}_1$ *et ii)* $\mathcal{M}_2 : \mathcal{D}_2 \rightarrow \mathcal{I}_2$*. deux modules différents. L'*Operator Sequential Execution *est défini par l'opérateur paramétrique :*

$$\longmapsto (\mathcal{M}_2, \mathcal{M}_1) : \mathcal{D}_1 \rightarrow \mathcal{I}_2$$

et l'opération $|\mathcal{M}_1 \longmapsto \mathcal{M}_2|$ *retourne un compound module comme résultat de l'exécution de* $\mathcal{M}_1$ *suivi de* $\mathcal{M}_2$*.*

L'opération $|\mathcal{M}_1 \longmapsto \mathcal{M}_2|$ *peut être effectué si et seulement si* $\mathcal{I}_1 \subseteq \mathcal{D}_2$*.*

L'opérateur présenté dans la définition 5 est un exemple d'opérateur ne supportant pas une exécution parallèle de ses *compound modules* impliqués, puisque l'entrée du second *compound module* est la sortie du premier.

L'opérateur suivant est utile pour exécuter des modules séquentiels créant des branchements de calcul selon une condition booléenne :

Definition 6 ***(Operator Conditional Sequential Execution)*** *Soient i)* $\mathcal{M}_1 : \mathcal{D}_1 \rightarrow \mathcal{I}_1$*, ii)* $\mathcal{M}_2 : \mathcal{D}_2 \rightarrow \mathcal{I}_2$ *et iii)* $\mathcal{M}_3 : \mathcal{D}_3 \rightarrow \mathcal{I}_3$ *et trois modules différents. L'*Operator Conditional Sequential Execution *est défini par l'opérateur paramétrique :*

$$\underset{<.>}{\longmapsto} (\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3) : \{0,1\} \times \mathcal{D}_1 \rightarrow \mathcal{I}_o$$

et l'opération $\left|\mathcal{M}_1 \underset{<cond>}{\longmapsto} \{\mathcal{M}_2, \mathcal{M}_3\}\right|$ *retourne un compound module comme résultat de l'exécution séquentiel de* $\mathcal{M}_1$ *suivi de* $\mathcal{M}_2$ *si* $< cond >$ *est* **vrai**, *ou de* $\mathcal{M}_3$ *le cas échéant.*

L'opération $\left|\mathcal{M}_1 \underset{<.>}{\longmapsto} \{\mathcal{M}_2, \mathcal{M}_3\}\right|$ *peut être appliquée si et seulement si* $\mathcal{I}_1 \subseteq \mathcal{D}_2 \cap \mathcal{D}_3$ *et* $\mathcal{I}_2 \cup \mathcal{I}_3 \subseteq \mathcal{I}_o$*.*

Nous pouvons exécuter séquentiellement des modules créant des boucles de calcul, en définissant les *compound modules* avec un autre opérateur conditionnel :

Definition 7 ***(Operator Cyclic Execution)*** *Soit* $\mathcal{M}_1 : \mathcal{D}_1 \rightarrow \mathcal{I}_1$ *un module. L'*Operator Cyclic Execution *est défini par :*

$$\circlearrowleft (\mathcal{M}_1) : \{0,1\} \times \mathcal{D}_1 \rightarrow \mathcal{I}_1$$

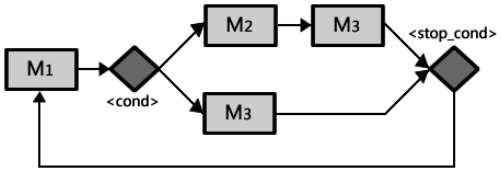


FIGURE 4

Algorithm 1: Code POSL de la figure 4

$$[\circlearrowleft < stop\_cond > \{ \left[\mathcal{M}_1 \underset{<cond>}{\longmapsto} \{\mathcal{M}_2; [\mathcal{M}_2 \longmapsto \mathcal{M}_3]\}\right] \}]$$

et l'opération $\circlearrowleft (< cond >) \{\mathcal{M}_1\}$ *retourne un compound module comme résultat de l'exécution séquentielle de* $\mathcal{M}_1$ *tant que* $< cond >$ *reste* **vrai**.

L'opération $\circlearrowleft (< cond >) \{\mathcal{M}_1\}$ *peut être exécutée si et seulement* $\mathcal{I}_1 \subseteq \mathcal{D}_1$*.*

Dans la figure 4, on présente un exemple simple combinant des *modules* utilisant les opérateurs de POSL introduis ci-dessus. L'algorithme 1 montre le code correspondant. Cet exemple montre trois *operation modules* faisant partie d'un *compound module* avec pour but de générer une configuration initiale à partir de laquelle débutera un solveur de recherche locale. Nous avons ainsi :

— $\mathcal{M}_1$, générant une configuration aléatoire.

— $\mathcal{M}_2$, sélectionnant une variable aléatoire d'une configuration donnée, et recopiant sa valeur dans une autre variable.

— $\mathcal{M}_3$, stockant la meilleure configuration trouvée.

Dans cet exemple, l'*operation module* $\mathcal{M}_2$ est d'abord exécuté, suivi de $\mathcal{M}_3$ si tous les éléments de la configuration générée sont différents (< `cond` >). Sinon, seul l'*operation module* $\mathcal{M}_3$ est exécuté. Cette opération est répétée un certain nombre de fois (< `stop_cond` >).

Jusque là, nous avons supposé que nous pouvons seulement exécuter deux modules, en utilisant la sortie du premier comme entrée du second. Seulement parfois, des modules peuvent avoir besoin d'autres informations en entrée provenant de *modules* exécutés bien avant eux. Dans ce cas, il est nécessaire de donner des entrées supplémentaires :

Definition 8 ***(Operator Cartesian Product)*** *Soient i)* $\mathcal{M}_1 : \mathcal{D}_1 \rightarrow \mathcal{I}_1$ *et ii)* $\mathcal{M}_2 : \mathcal{D}_2 \rightarrow \mathcal{I}_2$*. deux modules différents. L'*Operator Cartesian Product *est défini par l'opérateur paramétrique :*

$$\circledast (\mathcal{M}_1, \mathcal{M}_2) : \mathcal{D}_o \rightarrow \mathcal{I}_1 \times \mathcal{I}_2$$

et l'opération $\left|\mathcal{M}_1 \otimes \mathcal{M}_2\right|$ *retourne un compound module comme résultat de l'exécution de* $\mathcal{M}_1$ *et de* $\mathcal{M}_2$*. Son image est le produit cartésien de l'image des opérandes.*

L'opération $\left|\mathcal{M}_1 \otimes \mathcal{M}_2\right|$ *peut être appliquée si et seulement si* $\mathcal{D}_o \subseteq \mathcal{D}_1 \cap \mathcal{D}_2$*.*

La figure 5 montre un scénario où il est nécessaire d'utiliser cet opérateur.

POSL offre la possibilité de faire muter les solveurs. En fonction de l'opération, un ou plusieurs *module* opérande(s) sera exécutée(s), mais seule la sortie de l'un d'entre eux sera retournée par le *compound module*. Nous présentons ces opérateurs dans deux définitions, groupant ceux qui exécutent uniquement un opérande de *module* (définition 9) et ceux exécutant les deux opérandes (définition 10).

Definition 9 *Soient i)* $\mathcal{M}_1 : \mathcal{D}_1 \rightarrow \mathcal{I}_1$ *et ii)* $\mathcal{M}_2 : \mathcal{D}_2 \rightarrow \mathcal{I}_2$*. deux module différents et une probabilité* ρ*. Nous pouvons alors définir l'opérateur paramétrique suivant :*

1. $\otimes(\rho, \mathcal{M}_1, \mathcal{M}_2) : \mathcal{D}_o \rightarrow \mathcal{I}_o$ *Où :*

   $\left|\mathcal{M}_1 \otimes \mathcal{M}_2\right|$ *exécute* $\begin{cases} \mathcal{M}_1 & \text{avec probabilité } \rho \\ \mathcal{M}_2 & \text{avec probabilité } (1-\rho) \end{cases}$

2. $\otimes(\mathcal{M}_1, \mathcal{M}_2) : \mathcal{D}_o \rightarrow \mathcal{I}_o$ *Où :*

   $\left|\mathcal{M}_1 \otimes \mathcal{M}_2\right|$ *exécute* $\begin{cases} \mathcal{M}_1 & \text{si } \mathcal{M}_1 \text{ n'est pas } \textbf{null} \\ \mathcal{M}_2 & \text{sinon} \end{cases}$

Ces opérations peuvent être appliquée si et seulement si $\mathcal{D}_o \subseteq \mathcal{D}_1 \cap \mathcal{D}_2$ *et* $\mathcal{I}_1 \cap \mathcal{I}_2 \subseteq \mathcal{I}_o$ *sont vérifiés.*

La définition suivante fait appelle aux notions de *parallélisme coopératif* et de *parallélisme compétitif*. Nous disons qu'il y a parallélisme coopératif quand deux unités de calcul ou plus s'exécutent simultanément, et que le résultat obtenu provient de la combinaison des résultats calculés par chaque unité de calcul (voir définitions 10.1 et 10.2). À l'opposé, nous considérons qu'il y a parallélisme compétitif lorsque le résultat obtenu est une solution ne provenant que d'un seul processus exécuté en parallèle ; en général le premier processus à terminer (voir définition 10.3).

Definition 10 *Soient i)* $\mathcal{M}_1 : \mathcal{D}_1 \rightarrow \mathcal{I}_1$ *et ii)* $\mathcal{M}_2 : \mathcal{D}_2 \rightarrow \mathcal{I}_2$*. . deux module différents. Soient également* o_1 *et* o_2 *les sorties de* $\mathcal{M}_1$ *et* $\mathcal{M}_2$ *respectivement, telles qu'il existe une relation d'ordre totale entre elles. Nous définissons alors les opérateurs paramétriques suivants :*

1. $\oplus(\mathcal{M}_1, \mathcal{M}_2) : \mathcal{D}_o \rightarrow \mathcal{I}_o$ *où :*

   $\left|\mathcal{M}_1 \oplus \mathcal{M}_2\right|$ *retourne* $\max\{o_1, o_2\}$

2. $\ominus(\mathcal{M}_1, \mathcal{M}_2) : \mathcal{D}_o \rightarrow \mathcal{I}_o$ *où :*

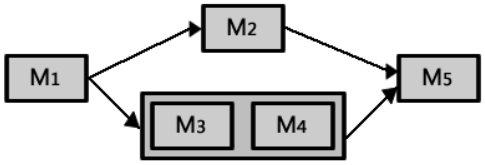


FIGURE 5

Algorithm 2: Code POSL de la figure 5

$$\mathcal{M}_1 \longmapsto \left[\!\!\left[\mathcal{M}_2 \otimes \left[\!\!\left[\mathcal{M}_3 \downarrow \mathcal{M}_4 \right]\!\!\right]_p \right]\!\!\right]_p \longmapsto \mathcal{M}_5$$

   $\left|\mathcal{M}_1 \ominus \mathcal{M}_2\right|$ *retourne* $\min\{o_1, o_2\}$

3. $\downarrow(\mathcal{M}_1, \mathcal{M}_2) : \mathcal{D}_o \rightarrow \mathcal{I}_o$ *où :*

   $\left|\mathcal{M}_1 \downarrow \mathcal{M}_2\right|$ *retourne*

   $\begin{cases} o_1 & \text{si } \mathcal{M}_1 \text{ termine en premier} \\ o_2 & \text{sinon} \end{cases}$

Ces trois opérations peuvent être appliquées si et seulement si $\mathcal{D}_o \subseteq \mathcal{D}_1 \cap \mathcal{D}_2$ *et* $\mathcal{I}_1 \cap \mathcal{I}_2 \subseteq \mathcal{I}_o$*.*

Nous illustrons un de ces trois opérateurs dans l'exemple de la figure 5. Les modules rentrant en jeu sont :

- $\mathcal{M}_1$, retournant une configuration
- $\mathcal{M}_2$, calculant un paramètre de tolérance ϵ
- $\mathcal{M}_3$ et $\mathcal{M}_4$, calculant le voisinage $\mathcal{N}$ de la configuration provenant de $\mathcal{M}_1$. Ces deux *operation modules* calcule $\mathcal{N}$ de manière différente, mais ils sont combinés de façon à ce que seul la sortie du premier module terminant son calcul soit retournée.
- $\mathcal{M}_5$, sélectionnant une configuration dans $\mathcal{N}$ améliorant le coût global avec un tolérance ϵ, à la manière du *Threshold Accepting Method* [1]

Dans l'algorithme 2, les *modules* $\mathcal{M}_3$ et $\mathcal{M}_4$ sont exécutés en parallèle, mais seule la sortie du premier module à terminer son calcul sera retournée.

De la même manière, le *compound module* composé de $\mathcal{M}_3$ et $\mathcal{M}_4$ peut être exécuté en parallèle avec $\mathcal{M}_2$, parce qu'ils sont indépendants l'un de l'autre et que l'opérateur le permet. Il est important de souligner qu'ils peuvent être exécutés par l'opérateur $\otimes$, puisqu'ils reçoivent la même entrée.

Les opérateurs introduits par les définitions 9 et 10 sont très utiles en terme de partage d'informations entre solveurs, mais également en terme de partage de comportements. Si un des opérandes est un *open channel* alors l'opérateur peut recevoir l'*operation module* d'un autre solveur, donnant la possibilité d'instancier ce module dans le solveur le réceptionnant.

L'opérateur va soit instancier le module s'il n'est pas *null* et l'exécuter, soit exécuter le module donné par le second opérande.

POSL contient également des opérateurs pour manipuler des *ensembles*, ce qui est particulièrement utile pour les solveurs basés sur des populations.

Definition 11 *Soient i)* $\mathcal{M}_1 : \mathcal{D}_1 \rightarrow 2^{\mathcal{I}_1}$ *et ii)* $\mathcal{M}_2 : \mathcal{D}_2 \rightarrow 2^{\mathcal{I}_2}$. *. deux modules différents, où* $2^{\mathcal{I}}$ *désigne l'ensemble des sous-ensembles d'un certain type de données* $\mathcal{I}$. *Soient les ensembles* V_1 *et* V_2 *les sorties respectives de* $\mathcal{M}_1$ *et* $\mathcal{M}_2$. *Nous définissons les opérateurs paramétriques suivants :*

1. $\textcircled{\cup}(\mathcal{M}_1, \mathcal{M}_2) : \mathcal{D}_o \rightarrow \mathcal{I}_o$ *où :*
   $\left|\mathcal{M}_1 \textcircled{\cup} \mathcal{M}_2\right|$ *retourne* $V_1 \cup V_2$
2. $\textcircled{\cap}(\mathcal{M}_1, \mathcal{M}_2) : \mathcal{D}_o \rightarrow \mathcal{I}_o$ *où :*
   $\left|\mathcal{M}_1 \textcircled{\cap} \mathcal{M}_2\right|$ *retourne* $V_1 \cap V_2$
3. $\textcircled{-}(\mathcal{M}_1, \mathcal{M}_2) : \mathcal{D}_o \rightarrow \mathcal{I}_o$ *où :*
   $\left|\mathcal{M}_1 \textcircled{-} \mathcal{M}_2\right|$ *retourne* $V_1 \backslash V_2$

Les opérateurs peuvent s'appliquer si et seulement on a $\mathcal{D}_o \subseteq \mathcal{D}_1 \cap \mathcal{D}_2$ *et* $\mathcal{I}_1 \cap \mathcal{I}_2 \subseteq \mathcal{I}_o$.

Un exemple classique d'*operation module* retournant des ensembles est l'algorithme calculant l'ensemble $\mathcal{N}_S$ des voisins d'une configuration donnée S dans une méthode de recherche locale. Prenons les *operation modules* suivants :

1. $\mathcal{M}_1$ qui, étant donnée une configuration S, calcule l'ensemble des configurations $\mathcal{N}_S^1 = \{S_i'\}$ où
   $$S_i' \in \mathcal{N}_S \Longleftrightarrow g(S_i, S) < \epsilon$$
   avec $\epsilon > 0$, et g une fonction de proximité.
2. $\mathcal{M}_2$ qui, étant donnée une configuration S, calcule l'ensemble des configurations $\mathcal{N}_S^2 = \{S_i'\}$ où chaque S_i' est obtenue en faisant varier la valeur de quelques variables de S choisies aléatoirement.

En partant des deux *operation modules* ci-dessus et en utilisant un des opérateurs présentés jusque-là, on peut créer un *compound module* pour calculer un voisinage de S étant orienté à la fois exploitation et exploration :

$$\ldots \longmapsto \left[\!\left[\mathcal{M}_1 \textcircled{\cup} \mathcal{M}_2\right]\!\right]_p \longmapsto \ldots$$

Dans ce cas, nous avons $\mathcal{N}_S = \mathcal{N}_S^1 \cup \mathcal{N}_S^2$.

Maintenant, nous définissons les opérateurs nous permettant d'envoyer de l'information vers d'autres solveurs. Deux types d'envois sont possibles :

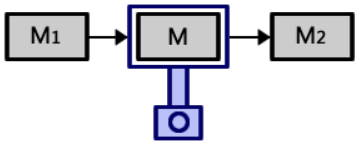


FIGURE 6

Algorithm 3: Code POSL de la figure 6 cas (a)
$\mathcal{M}_1 \longmapsto (\!\|\mathcal{M}\|\!)^o \longmapsto \mathcal{M}_2$

(a) on exécute un *operation module* et on envoie sa sortie,

(b) ou on envoie l'*operation module* lui-même.

Definition 12 *Soit* $\mathcal{M} : \mathcal{D} \rightarrow \mathcal{I}$ *un operation module. Nous avons les opérateurs paramétriques suivant :*

1. $(\!|.|\!)^o(\mathcal{M}) : \mathcal{D} \rightarrow \mathcal{I}$ *où :*
   $(\!|\mathcal{M}|\!)^o$ *exécute* $\mathcal{M}$ *et envoie la sortie*
2. $(\!|.|\!)^m(\mathcal{M}) : \mathcal{D} \rightarrow \mathcal{I}$ *où :*
   $(\!|\mathcal{M}|\!)^m$ *exécute* $\mathcal{M}$ *mais envoie l'operation module plutôt que sa sortie.*

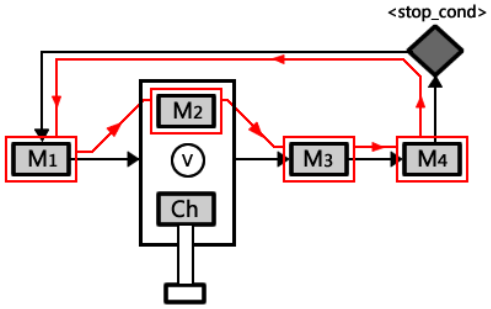
Algorithm 4: Code POSL de la figure 6 cas (b)
$\mathcal{M}_1 \longmapsto (\!\|\mathcal{M}\|\!)^m \longmapsto \mathcal{M}_2$

Les algorithmes 3 et 4 montrent du code utilisant POSL illustrant deux situations possibles de la configuration de la figure 6 : a) envoyer le résultat de l'exécution de l'*operation module* $\mathcal{M}$ b) envoyer l'*operation module* $\mathcal{M}$ lui-même.

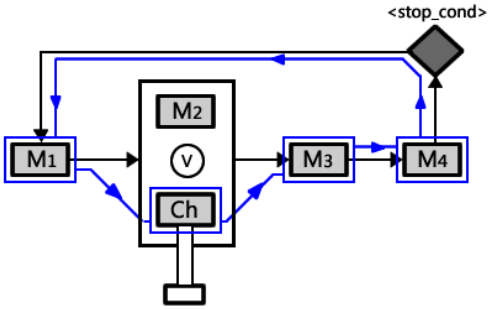
La figure 7 montre un autre exemple où l'on peut combiner un *open channel* avec l'*operation module* $\mathcal{M}_2$ à travers l'opérateur $\textcircled{\vee}$. L'*operation module* $\mathcal{M}_2$ sera exécuté tant que l'*open channel* reste **null** (figure 7a, lignes rouges). Si un *operation module* a été reçu par l'*open channel*, il sera exécuté à la place de l'*operation module* $\mathcal{M}_2$ (figure 7b lignes bleues).

Avec les opérateurs présentés jusqu'ici, nous sommes en mesure de concevoir des stratégies (ou algorithme) de résolution d'un problème d'optimisation combinatoire donné. Une fois une telle stratégie définie, on peut changer les composants (*operation module* et *open channel*) auxquels elle fait appel, permettant ainsi d'implémenter différents solveurs à partir de la même stratégie mais composés de différents modules, du moment que ces derniers respectent la signature attendue, à savoir le typage des entrées et sorties.

Pour définir une *computation strategy*, nous utiliserons l'environnement suivant :



(a) Le solveur exécute son propre *operation module*



(b) Le solveur exécute l'*operation module* provenant d'un autre solveur

FIGURE 7 – Deux comportements différents dans le même solveur

```
St := cStrategy
oModule: < liste des types d'operation modules > ;
oChannel: < liste des types d'open channels > ;
{
        < ...computation strategy... >
}
```

Avant de programmer la *computation strategy*, il est nécessaire de déclarer les types d'*operation modules* et d'*open channels* qui seront utilisés. Ceci est spécifié par `< liste des types d'operation modules >` et `< liste des types d'open channels>` respectivement. Après ces déclarations, le corps du code contient des opérateurs de POSL combinant des *module*. Un exemple simple est illustré par l'algorithme 5.

3.4 Définition de solveur

Une fois des *operation modules*, *open channels* et *computation strategy* définis, nous pouvons créer des solveurs en choisissant et instanciant les composants que nous allons intégrer au solveur. POSL fournit un environnement pour cela :

```
solver_k := solver
{
cStrategy: < strategy > ;
oModule: < liste des instances d'operation modules > ;
oChannel: < liste des instances d'open channels > ;
}
```

3.5 Définition des communications

Une fois que nous avons défini la stratégie de nos solveurs, l'étape suivante est de déclarer les canaux de communication reliant certains solveurs entre eux. Jusque là, les solveurs sont effectivement déconnectés, mais ont tout pour établir des communications (voir figure 1c). POSL propose à l'utilisateur une plate-forme pour facilement définir une *méta-strategie* à suivre par l'ensemble des solveurs.

Les communications sont établies en respectant les règles suivantes :

1. À chaque fois qu'un solveur envoie une information via les opérateurs $(\!|.|\!)^o$ ou $(\!|.|\!)^m$, il créé une *prise mâle de communication*
2. À chaque fois qu'un solveur contient un *open channel*, il créé une *prise femelle de communication*
3. Les solveurs peuvent être connectés entre eux en créant des *subscriptions*, reliant *prises mâles* et *femelles* (voir figure 8).

Avec l'opérateur $(\cdot)$, nous pouvons avoir accès aux *operation modules* envoyant une information et aux noms des *open channels* d'un solveur. Par exemple : $Solver_1 \cdot \mathcal{M}_1$ fournit un accès à l'*operation module* $\mathcal{M}_1$ du $Solver_1$ si et seulement si il est utilisé par l'opérateur $(\!|.|\!)^o$ (ou $(\!|.|\!)^m$), et $Solver_2 \cdot Ch_2$ fournit un accès à l'*open channel* Ch_2 de $Solver_2$. Nous définissons à présent les *subscriptions*.

Definition 13 *Supposons deux solveurs différents* $Solver_1$ *et* $Solver_2$*. Nous pouvons les connecter grâce à l'opération suivante :*

$$Solver_1 \cdot \mathcal{M}_1 \rightsquigarrow Solver_2 \cdot Ch_2$$

La connexion peut être définie si et seulement si :

1. $Solver_1$ *possède un operation module appelé* $\mathcal{M}_1$ *encapsulé dans l'opérateur* $(\!|.|\!)^o$ *ou* $(\!|.|\!)^m$*.*
2. $Solver_2$ *contient un open channel appelé* Ch_2 *recevant le même type d'informations envoyées par* $\mathcal{M}_1$*.*

La définition 13 n'exprime que la possibilité de définir statiquement des stratégies de communication. À terme, nous aimerions inclure dans POSL des opérateurs permettant plus de souplesse et d'expressivité en terme de communication entre solveurs, notamment à travers des stratégies dynamiques de communication.

4 Un solveur POSL

Dans cette section, nous allons expliquer la structure d'un solveur créé par POSL en nous appuyant sur

un exemple. Nous choisissons un des algorithmes les plus classiques : une méta-heuristique de recherche locale. Les méta-heuristiques ont une structure commune : ils commencent par initialiser quelques structures de données qui leur sont propres (une liste tabou pour le *Tabu Search* [7], une température pour le *Simulated Annealing* [10], etc). Puis, une configuration initiale s est générée, soit aléatoirement soit à travers une certaine heuristique. Viennent ensuite les étapes de résolution à proprement parlé, à savoir la sélection d'une nouvelle configuration $s*$ extraite parmi le voisinage $\mathcal{V}(s)$ de la configuration actuelle. Si $s*$ est une solution au problème $\mathcal{P}$ alors le processus s'arrête et $s*$ est retournée, sinon les structures de données sont actualisées, $s*$ est prise en compte ou non pour une nouvelle itération, en fonction de certains critères (comme par exemple la pénalisation d'optimum locaux du *Guided Local Search* [1]).

Les *restarts* constituent un mécanisme classique pour éviter de rester piégé dans un minimum local. Ils sont déclenchés lorsque l'on n'arrive plus à trouver une meilleure configuration, ou parfois après un *timeout*.

Les *operation modules* composant un algorithme de recherche locale sont décrits ci-dessous :

$O.\ M. - 1:$	Génére une configuration s
$O.\ M. - 2:$	Défini le voisinage $\mathcal{V}(S)$
$O.\ M. - 3:$	Sélectionne $s* \in \mathcal{V}(s)$
$O.\ M. - 4:$	Évalue le critère d'acceptation pour $s*$

Nous pouvons combiner les modules pour en créer de plus complexes. Par exemple, si l'on souhaite un *operation module* générant une configuration aléatoire mais capable de donner parfois une configuration prédéfinie (par exemple passée en paramètre) suivant une certaine probabilité, notre *operation module* pourrait être la combinaison via l'opérateur (ρ) de deux modules basiques s'occupant respectivement de générer une configuration aléatoire et de renvoyer une configuration donnée en paramètre.

Complexifions encore un peu notre solveur : supposons que nous avons une fonction de voisinage qui est orientée exploitation, mais que parfois nous souhaitons demander à d'autres solveurs de nous envoyer leur *operation module* calculant le voisignage, afin d'exécuter l'un de ces derniers dans notre solveur. Ceci est modélisable par l'*open channel* suivant :

$O.\ Ch. - 1:$	Demande de l'*operation module* $\mathcal{V}(s)$.

Supposons que nous aimerions toujours communiquer à tout le monde notre configuration courante. Il est possible d'appliquer l'opérateur $(\!|.|\!)^o$ à l'*operation module* 4.

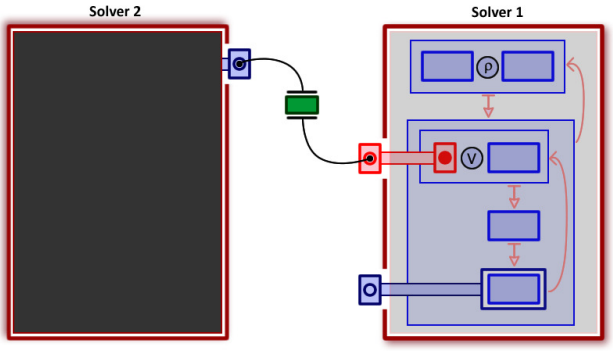


FIGURE 8 – Stratégie de coopération entre solveurs

Algorithm 5: Stratégie générale d'une méta–heuristique de recherche locale

```
St ← strategy
oModule : M_1^a, M_1^b, M_2, M_2, M_4 ;
oChannel : Ch_1 ;
{
    [↻ (Iterations % 10000 != 0) {
        [M_1^a (ρ) M_1^b] ⟼
        [↻ (Iterations % 1000 != 0) {
            [Ch_1 (∨) M_2] ⟼ M_3 ⟼ (|M_4|)^o
        }]
    }]
}
```

La figure 8 présente l'exemple ci-dessus. L'*open channel* de *Solver-1* est représenté en rouge. *Solver-1* contient un *open channel* **demandant** une fonction de voisinage. Il peut donc être connecté à *Solver-2* puisque ce dernier applique l'opérateur $(\!|.|\!)^o$ pour **envoyer** son *operation module* de voisinage. Une *subscription* (bloc vert) est alors créée définissant le canal de communication. La figure montre qu'aucun solveur ne contient d'*operation module* permettant d'envoyer une configuration, laissant ainsi le module chargé d'évaluer une configuration sans communication avec l'extérieur.

L'algorithme 5 montre un code POSL pour la *computation strategy* du solveur *Solver-1* décrit précédemment.

L'algorithme 6 montre une définition de solver. Nous mettons ici en place une *computation strategy*, suivit des instances des *modules* (*operation modules* et *open channels*).

Supposons qu'il existe un autre solver Σ_2 avec un *operation module* $\mathcal{M}_{\mathcal{V}}$ partageant sa fonction de voisinage. Nous pourrions le connecter avec le solveur Σ_1 comme le montre l'algorithme 7.

Algorithm 6: Definition d'un solveur utilisant une méta–heuristique de recherche locale

$\Sigma_1 \leftarrow$ **solver**
{
strategy : St
oModule : m_1^a, m_1^b, m_2, m_3, m_4
oChannel : ch_1
}

Algorithm 7: Définition de la communication entre solveurs

$\Sigma_2 \cdot \mathcal{M}_{\mathcal{V}} \rightsquigarrow \Sigma_1 \cdot Ch_1$

5 Conclusion

Dans ce papier, nous avons présenté POSL, un système pour construire des solveurs parallèles. Il propose une manière modulable pour créer des solveurs capables d'échanger n'importe quel type d'informations, comme par exemple leur comportement même, en partageant leurs *operation modules*. Avec POSL, de nombreux solveurs différents pourront être créés et lancés en parallèle, en utilisant une unique stratégie générique mais en instanciant différents *operation modules* et *open channels* pour chaque solveur.

Il est possible d'implémenter différentes stratégies de communication, puisque POSL fournit une couche pour définir les canaux de communication connectant les solveurs entre eux via des *subscriptions*.

L'implémentation de POSL reste un travail en cours. Notre principale tâche est de créer un système aussi général et souple que possible, permettant d'inclure de nouvelles fonctionnalités dans un futur proche. Notre but est d'obtenir une bibliothèque riche en *operation modules* et *open channels* à proposer aux utilisateurs, basés sur une étude approfondie des algorithmes classiques de résolution de problèmes combinatoires. Ainsi, nous espérons faciliter et accélérer la conception de nouveaux algorithmes.

En parallèle, nous prévoyons de développer de nouveaux opérateurs, en fonction des besoins des développeurs. Il est nécessaire, par exemple, d'améliorer la langage de définition de solveurs afin d'accélérer et simplifier encore le processus de production de solveurs. En outre, nous visons à étendre le langage de définition de communication pour permettre de créer de plus complexes et versatiles stratégies de communication, utiles pour étudier le comportement des solveurs parallèles.

À moyen terme, nous sommes intéressés par l'intégration de méthodes d'apprentissage automatique afin de permettre aux solveurs de s'adapter automatiquement, en fonction par exemple des résultats des solveurs qui leur sont voisins dans le réseau.

Références

[1] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237 :82–117, July 2013.

[2] Alexander E.I. Brownlee, Jerry Swan, Ender Özcan, and Andrew J. Parkes. Hyperion 2. A toolkit for {meta- , hyper-} heuristic research. Technical report, 2014.

[3] Daniel Diaz, Florian Richoux, Philippe Codognet, Yves Caniou, and Salvador Abreu. Constraint-Based Local Search for the Costas Array Problem. In *Learning and Intelligent Optimization*, pages 378–383. Springer Berlin Heidelberg, 2012.

[4] Talbi El-Ghazali. Combining metaheuristics with mathematical programming, constraint programming and machine learning. *4or*, July 2013.

[5] Stephan Frank, Petra Hofstedt, and Pierre R. Mai. Meta-S : A Strategy-Oriented Meta-Solver Framework. In *FLAIRS Conference*, 2003.

[6] Alex S Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary computation*, 16(1) :31–61, January 2008.

[7] Michel Gendreau and Jean-Yves Potvin. Tabu Search. In *Handbook of Metaheuristics*, pages 41–59. Springer US, 2010.

[8] Youssef Hamadi, Said Jaddour, and Lakhdar Sais. Control-Based Clause Sharing in Parallel SAT Solving. In *Autonomous Search*, pages 245–267. Springer Berlin Heidelberg, 2012.

[9] Danny Munera, Daniel Diaz, Salvador Abreu, and Philippe Codognet. A Parametric Framework for Cooperative Parallel Local Search. In *14th European Conference, EvoCOP*, pages 13–24, Granada, 2014.

[10] Alexander G. Nikolaev and Sheldon H. Jacobson. Simulated Annealing. In *Handbook of Metaheuristics*, pages 1–39. Springer US, 2010.

[11] Brice Pajot and Eric Monfroy. Separating Search and Strategy in Solver Cooperations. In *Perspectives of System Informatics*, pages 401–414. Springer Berlin Heidelberg, 2003.

[12] Jerry Swan and Nathan Burles. Templar - a framework for template-method hyper-heuristics. *Lecture Notes in Computer Science*, 9025 :205–216, 2015.

Une approche basée sur la programmation stochastique multi-étapes pour résoudre le VRPTW dynamique et stochastique

Michael Saint-Guillain[1] **Yves Deville**[1] **Christine Solnon**[2]

[1]ICTEAM, Université catholique de Louvain, Belgium
[2]Université de Lyon, CNRS
[2]INSA-Lyon, LIRIS, UMR5205, F-69621, France
{michael.saint,yves.deville}@uclouvain.be christine.solnon@liris.cnrs.fr

Résumé

Ce papier résume l'article [3] accepté à la conférence « CPAIOR2015 ». Nous considérons un problème dynamique de tournées de véhicules avec fenêtres temporelles (DS-VRPTW), dans lequel les clients sont libres de soumettre leurs requêtes alors que les véhicules ont déjà commencé leurs tournées. Ces clients sont dits « stochastiques » car nous disposons d'information stochastique (en termes de distributions de probabilités) sur les requêtes futures. Nous proposons une nouvelle règle de décision, GSA, ainsi qu'une approche heuristique visant à l'approximer. Nous introduisons également une nouvelle stratégie d'attente. Les résultats expérimentaux montrent que notre approche est compétitive avec l'état de l'art.

Abstract

This paper summarizes paper [3] accepted to the « CPAIOR2015 » conference. We consider a dynamic vehicle routing problem with time windows and stochastic customers (DS-VRPTW), such that customers may request for services as vehicles have already started their tours. We introduce a new decision rule, GSA, as well as a heuristic approach for efficiently approximating it. We introduce a new waiting strategy. Experiments show that our approach is competitive with state-of-the-art methods.

1 Description du problème

1.1 Données du DS-VRPTW

Considérons un horizon de temps discret $[1, H]$, de façon à ce que chaque évènement aléatoire ou décision soit associé à une unité discrète de temps $t \in [1, H]$. Le DS-VRPTW est défini sur un graphe complet de n sommets représentant les *positions potentielles de requêtes* (PPR). Les arrêtes sont pondérées par les temps de trajets séparant les sommets. À chaque PPR sont associés une quantité demandée, un temps requis pour servir la requête ainsi qu'une fenêtre temporelle.

On note $R \subseteq [1, n] \times [0, H]$ l'ensemble des requêtes. Chaque requête (i, t) se caractérise donc par le PPR associé i et par l'unité de temps t à laquelle elle est révélée. Si $t = 0$, la requête est connue avant l'exécution du problème et celle-ci est dite *déterministe*. Si $t > 0$, la requête est révélée durant l'exécution du problème, c'est-à-dire alors que les véhicules ont déjà commencé leurs tournées ; elle est dite *dynamique*. Durant l'exécution du problème et à un temps donné $t \in [1, H]$, seules les requêtes déjà révélées sont connues. Cependant, à chaque unité de temps à venir $t' \in [t+1, H]$ est associé un vecteur de probabilités $P^{t'}$ décrivant, pour chaque sommet $i \in [1, n]$, la probabilité $P^{t'}[i]$ qu'une requête soit révélée pour le sommet i au temps t'.

Afin de satisfaire les requêtes, k véhicules de même capacité sont disponibles.

1.2 Solution du DS-VRPTW

À la fin de l'horizon de temps, une solution est constituée d'un sous-ensemble des requêtes $R_a \subseteq R$, ainsi que de k routes (une pour chaque véhicule). R_a contient les requêtes ayant été *acceptées* ; les requêtes dans $R \setminus R_a$ ont été *rejetées*. Les routes obtenues à la fin de l'horizon doivent impérativement satisfaire les contraintes du VRPTW classique, restreintes aux seules requêtes R_a.

Une solution au DS-VRPTW est construite durant l'exécution. À chaque unité de temps $t \in [1, H]$, une action a^t est calculée, composée de deux parties : pour chaque requête révélée au temps t, une décision d'acceptation ou de refus ; pour chaque véhicule, les décisions opérationnelles à appliquer au temps t (servir une requête, voyager vers un sommet ou attendre au sommet courant). Certaines décisions peuvent également être prises à $t = 0$. Une solution est donc une séquence d'actions $a^{0..H}$ couvrant l'ensemble de l'horizon de temps.

L'objectif est de minimiser, à la fin de l'horizon de temps, le nombre total de requêtes rejetées par $a^{0..H}$.

2 Résolution du problème

Afin de résoudre le DS-VRPTW, une règle de décision doit choisir, à chaque unité de temps, la prochaine action à effectuer étant données les actions précédentes, les requêtes déjà connues ainsi que l'information stochastique sur les requêtes futures.

La meilleure action à un temps donné $t \in [1, H]$ est celle qui minimise l'*espérance du nombre de requêtes rejetées, sur le restant de l'horizon* $[t+1, H]$. Ce type de modélisation est appelé *programme stochastique multi-étapes* (*multistage stochastic program*).

Bien que fournissant une action optimale, la résolution d'un programme stochastique multi-étapes est beaucoup trop couteuse pour pouvoir être appliquée à des situations réalistes. Par conséquent, les règles de décision conçues pour résoudre les DS-VRPTW préfèrent généralement approximer le programme stochastique plutôt que le résoudre optimalement.

2.1 Une nouvelle règle de décision

Nous proposons une nouvelle règle de décision appelée Global Stochastic Assessment (GSA), que nous comparons avec les règles existantes comme Multiple Scenario Approach (MSA) [2]. GSA est basée sur la génération de scénarios.

Contrairement à MSA, GSA a la particularité de respecter les contraintes de non-anticipation et par là-même de prendre de meilleures décisions dans notre contexte stochastique. En d'autres termes, alors que les approches de type MSA approximent un programme stochastique en deux étapes, GSA approxime un programme stochastique multi-étapes.

2.2 Une nouvelle stratégie d'attente

Un véhicule terminant le service d'une requête a généralement le choix entre voyager directement vers un autre sommet du graphe ou attendre à sa position courante. Bien qu'inutiles dans un contexte deterministe, les temps d'attente peuvent cependant s'avérer judicieux lorsque des requêtes ont la possibilité d'être révélées dynamiquement durant l'exécution.

Par ailleurs, les stratégies de repositionnement permettant à un véhicule de se déplacer vers un sommet pour lequel aucune requête n'est (encore) connue apportent également davantage d'anticipation [1]. La nouvelle stratégie d'attente proposée dans [3] permet de déterminer heuristiquement les temps d'attente le long d'une route. Cette nouvelle stratégie permet en outre de tirer un meilleur profit des sommets de repositionnement.

2.3 Algorithme heuristique

Un algorithme permettant de résoudre le DS-VRPTW est décrit dans [3]. Basé sur la génération de scénarios et la recherche locale, l'algorithme intègre la règle de décision GSA afin de calculer dynamiquement les actions à opérer durant l'exécution du problème.

3 Expérimentations

Les expériences réalisées sur des instances de test, incluant différents degrés de dynamisme, montrent que non seulement notre nouvelle approche est compétitive par rapport aux méthodes de l'état de l'art, mais aussi que celle-ci permet le calcul a priori de solutions anticipatives utiles pour des problèmes totalement dynamiques, c'est-à-dire dans lesquels aucune requête n'est connue avant de commencer les tournées. De plus, les résultats montrent que lorsque le degré de dynamisme augmente, la nouvelle stratégie d'attente proposée tend à produire de meilleurs résultats que les stratégies existantes.

Références

[1] Russell Bent and Pascal Van Hentenryck. Waiting and Relocation Strategies in Online Stochastic Vehicle Routing. *IJCAI*, pages 1816–1821, 2007.

[2] Russell W Bent and Pascal Van Hentenryck. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(6) :977–987, 2004.

[3] Michael Saint-Guillain, Yves Deville and Christine Solnon. A Multistage Stochastic Programming Approach to the Dynamic and Stochastic VRPTW. *Accepted for publication in conference CPAIOR*, 2015. `http://becool.info.ucl.ac.be/pub/papers/multistage_vrptw_extended_0.pdf`

Un algorithme arborescent de contraction forte pour le CSP numérique

Olivier Sans **Remi Coletta** **Gilles Trombettoni**

LIRMM, Université de Montpellier, 161 rue Ada, 34095 Montpellier, France
{prenom.nom}@lirmm.fr

Résumé

Nous proposons dans cet article un nouvel opérateur de contraction forte pour le CSP numérique. Cet opérateur, nommé TEC (*Tree for Enforcing Consistency*), construit un sous-arbre d'exploration avant d'effectuer l'union (l'enveloppe) des espaces de recherche associés aux feuilles de ce sous-arbre. TEC construit le sous-arbre en largeur d'abord par un processus de bissection et de contraction. TEC généralise à plusieurs dimensions le principe de la disjonction constructive utilisée par l'opérateur CID existant. A la différence des opérateurs classiques de contraction forte, tels que CID ou 3B, qui considèrent toutes les variables du problème, TEC concentre son travail sur un sous-ensemble de variables à bissecter de manière adaptative. L'enveloppe des boîtes feuilles du sous-arbre TEC étant contrainte à être parallèle aux axes, la boîte résultante surestime l'espace de recherche. Pour éviter cet inconvénient, nous proposons Graham-TEC, qui permet d'inférer des contraintes implicites linéaires. Graham-TEC génère une enveloppe convexe avec des demi-plans portant sur des paires de variables. Des expériences préliminaires portant sur un banc d'essais de l'état de l'art valident l'intérêt de notre approche, en montrant des gains significatifs sur plusieurs instances difficiles.

1 Introduction

Les solveurs par intervalles permettent de résoudre des systèmes de contraintes numériques (c'est-à-dire des équations ou inégalités à valeurs dans $\mathbb{R}$) en prenant en compte des incertitudes (bornées) dans les paramètres (dues à des erreurs de mesure en physique par exemple) et les erreurs d'arrondis des calculs sur les nombres flottants. Des calculs rigoureux sont rendus possibles par la manipulation de domaines des variables bornés par des nombres flottants et l'utilisation de l'arithmétique d'intervalles.

Le processus de résolution est basé sur une stratégie de branchement et de contraction (*Branch & Contract*). Le branchement consiste à sélectionner une variable du système et à couper son domaine en (généralement) deux sous-intervalles. Ce processus génère un arbre d'exploration. La contraction (ou filtrage) consiste à réduire les domaines des variables du système sans perte de solution afin de limiter l'explosion combinatoire. Les opérateurs de contraction existants proviennent de trois domaines de recherche : la programmation par contraintes, l'analyse par intervalles et la programmation mathématique. Le processus de branchement et contraction enchaîne couramment un contracteur de programmation par contraintes et un opérateur de convexification polyhédrale.

En programmation par contraintes sur domaines finis, un principe de rognage (*shaving*) peut être utilisé pour filtrer les domaines. L'algorithme établissant la *singleton arc-consistency* (SAC [8]) affecte temporairement une valeur à une variable (les autres étant temporairement éliminées) et calcule une cohérence locale sur le sous-problème correspondant. Si une incohérence est obtenue, la valeur est supprimée du domaine dans tout le sous-arbre, sinon elle est maintenue. Dans le dernier cas, la réduction effectuée sur les domaines des autres variables est perdue. La disjonction constructive permet de mieux exploiter cette réduction en effectuant l'union des sous-problèmes générés par l'affectation de chaque valeur à la variable. Ce principe, mis en œuvre dans l'algorithme Partition-1-AC [6, 4], produit une cohérence strictement plus forte que la SAC. Elle donne un espoir de réduction sur potentiellement toutes les variables du système quand l'algorithme travaille sur (rogne) une seule variable.

Sur les domaines continus qui contiennent une infinité de nombres réels, les intervalles sont divisés en un nombre spécifié de sous-intervalles qui jouent le rôle de valeurs dans le CSP sur domaines finis. L'opérateur de contraction CID [19] applique ainsi le principe

de la disjonction constructive sur chaque variable du système prise une à une. Il déclenche un algorithme de propagation de contraintes, comme 2B [11, 12, 5] ou Mohc [2], sur chaque sous-problème correspondant à un sous-intervalle de la variable traitée. L'union des différents domaines contractés est alors retournée. Plus précisément, la boîte (produit cartésien des intervalles) *enveloppe* de cette union est rendue car les intervalles sont réduits seulement aux bornes (pas de *trous* au milieu des intervalles).

Dans cet article, nous proposons un nouvel opérateur de contraction, appelé TEC (*Tree for Enforcing Consistency*), qui généralise l'opérateur CID en appliquant le rognage sur plusieurs variables simultanément. TEC effectue un parcours arborescent de type *Branch & Contract* dont le nombre de nœuds est borné par un paramètre donné. La contraction est obtenue en retournant la boîte enveloppe des feuilles de cet arbre, c'est-à-dire la plus petite boîte (parallèle aux axes) qui englobe toutes les boîtes feuilles.

La boîte enveloppe, qui peut se voir comme un ensemble de contraintes (linéaires) unaires sur les variables du système, peut grandement surestimer l'union des boîtes feuilles. C'est pourquoi nous proposons une variante de TEC, appelée Graham-TEC, qui infère des contraintes linéaires implicites pour chaque paire de variables. Une enveloppe convexe de la projection des boîtes feuilles en deux dimensions permet de mieux approximer l'union des feuilles. Ces contraintes binaires linéaires sont ajoutées au polytope généré par l'opérateur X-Newton [3] de convexification polyhédrale utilisé pendant la résolution pour mieux contracter les domaines.

Après les définitions et les notations présentées à la section 2, la section 3 décrit l'opérateur TEC et sa mise au point expérimentale. La section 4 présente Graham-TEC et la section 5 détaille des expériences préliminaires encourageantes obtenues par une stratégie enchaînant Graham-TEC et l'opérateur X-Newton enrichi par les contraintes linéaires fournies par Graham-TEC.

2 Définitions et notations

Les algorithmes présentés dans cet article permettent de résoudre des systèmes ou réseaux de contraintes numériques. Un **réseau de contraintes numériques** est défini par un triplet $P = (X, [X], C)$, où X représente un ensemble de n variables réelles à valeurs dans $[X]$. Nous notons $[x_i] = [\underline{x_i}, \overline{x_i}]$, l'intervalle/domaine de la variable $x_i \in X$, où $\underline{x_i}$ et $\overline{x_i}$ sont des nombres flottants. Une contrainte de C est une équation ou une inégalité sur les nombres réels définie par une expression mathématique classique pouvant contenir des opérateurs arithmétiques, trigonométriques, d'exponentiation, etc. Une solution de P est un vecteur n-dimensionnel de $[X]$ satisfaisant l'ensemble des contraintes de C.

Le domaine $[X] = [x_1] \times ... \times [x_n]$ est un produit cartésien d'intervalles communément nommé boîte (parallèle aux axes). La **largeur** ou **taille** d'un intervalle $[x_i]$ est définie par $\omega(x_i) = \overline{x_i} - \underline{x_i}$. La largeur d'une boîte est donnée par la plus grande largeur sur l'ensemble des intervalles de la boîte, c'est-à-dire : $\max_{i=1..n}\ \omega(x_i)$.

L'union de plusieurs boîtes n'étant généralement pas une boîte, un opérateur $Hull$ d'enveloppe a été défini.

Définition 1. *Soit $box_1, ..., box_m$ un ensemble de boîtes n-dimensionnelles. L'opération* ***Hull*** *de $box_1, ..., box_m$, notée $Hull(box_1, ..., box_m)$, est la boîte minimale incluant l'ensemble de ces m boîtes.*

Un réseau de contraintes numériques peut être résolu par une stratégie de branchement et de contraction (*Branch & Contract*) mentionnée en introduction. Ce processus, détaillé à l'algorithme 1, débute avec le domaine initial $box = [X]$ et se termine lorsque la taille des boîtes feuilles de l'arbre d'exploration est inférieure à une précision ϵ donnée. Le parcours est effectué en profondeur d'abord. Il est garanti que les boîtes éliminées par contraction ne contiennent pas de solution réelle. Les boîtes feuilles rangées dans *AtomicBoxes* puis renvoyées par l'algorithme peuvent contenir ou non des solutions. Des algorithmes comme celui de *Newton sur intervalles* [13] peuvent être utilisées pour essayer de garantir l'existence d'une solution réelle dans les boîtes atomiques.

Algorithm 1: `Branch&Contract`

```
   Input: N = (X, box, C) : an NCN, ε : precision parameter
   Output: A set of atomic boxes including all the solutions
1  Boxes ← {box}                          /* stack (LIFO) of boxes */
2  AtomicBoxes ← ∅                        /* set of atomic boxes */
3  while Boxes ≠ ∅ do
4      box' ← Boxes.pop()
5      box' ← CPContract(box')
6      box' ← ConvexifyContract(box')
7      if box' ≠ ∅ then
8          if ω(box') < ε then
9              AtomicBoxes.push(box')
10         else
11             (box1, box2) ← Bisect(box')
12             Boxes.push(box1)
13             Boxes.push(box2)
14 return AtomicBoxes
```

La **contraction** (ou filtrage) se définit informellement par la réduction des domaines aux bornes sans perte de solution. L'algorithme 1 souligne que la plupart des solveurs à intervalles font appel à deux opérateurs de filtrage en séquence, l'un issu de la programmation par contraintes ($CPcontract$) et l'autre de l'analyse par intervalles ou de la programmation mathématique. Ce dernier ($ConvexifyContract$) se

base sur une convexification polyhédrale de l'espace-solution pour contracter les domaines.

Plusieurs opérateurs de contraction ont été proposés par la communauté de programmation par contraintes. L'opérateur HC4 [5, 12] permet de calculer la 2B-cohérence [11]. Cette cohérence locale s'apparente à la cohérence de bornes (*Bound-consistency*), une forme d'arc-cohérence restreinte aux bornes des intervalles.

Comme expliqué en introduction et de manière similaire à SAC pour le CSP sur domaines finis [8], l'algorithme établissant la 3B-cohérence [11] se base sur un principe de rognage qui élimine un sous-intervalle aux bornes des domaines quand l'appel à un algorithme comme la 2B-cohérence sur le sous-problème correspondant produit un domaine vide (prouvant l'absence de solution dans le sous-problème). L'opérateur CID (*Constructive Interval Disjunction*) se base sur le principe de la disjonction constructive appliqué aux domaines des variables. La procédure principale de CID, VarCID, sélectionne une variable x_i. L'intervalle $[x_i]$ est découpé en s_{cid} sous-intervalles ; chaque sous-problème correspondant est contracté par un opérateur de contraction tel que HC4, puis le $Hull$ de l'ensemble des boîtes résultantes est retourné. Ce processus est itéré sur chaque variable tant qu'une contraction est obtenue.

3 L'opérateur TEC

La disjonction constructive de l'opérateur CID peut produire une contraction sur l'ensemble du réseau lorsqu'un rognage est effectué sur une seule variable. L'opérateur TEC généralise ce principe en ne rognant qu'un sous-ensemble des variables mais de façon simultanée. TEC construit en fait un sous-arbre d'exploration de taille bornée avant d'effectuer le *Hull* des boîtes feuilles générées. En chaque nœud du *sous-arbre TEC*, est appliquée une cohérence locale telle que la 2B-cohérence. L'algorithme 2 détaille le fonctionnement de TEC.

3.1 Algorithme

TEC met en œuvre un processus de bissection et de contraction en construisant en largeur d'abord un sous-arbre de taille bornée. Ce processus combinatoire (recherche arborescente) est réalisé par la boucle `while` (ligne 4). La boîte initiale est tout d'abord contractée par le sous-contracteur Φ avant d'initialiser la file $Boxes$. A chaque itération, la première boîte de $Boxes$ est extraite selon un principe *First In First Out* mettant en œuvre le parcours en largeur d'abord. Si cette boîte n'est pas vide (non détectée incohérente par la cohérence locale) et si sa taille est supérieure à un paramètre de precision ϵ, elle est bissectée sur une dimension et chacune des sous-boîtes générées est alors contractée par Φ puis placée dans $Boxes$. Si la boîte est trop petite (de taille inférieure à ϵ), elle est ajoutée dans $AtomicBoxes$. Ce processus s'arrête lorsque le parcours arborescent a contracté un nombre de noeuds maximum (paramètre $TECnodes$ [1]) ou lorsque la file $Boxes$ est vide. L'enveloppe ($Hull$) de l'ensemble des boîtes feuilles ($Boxes \cup AtomicBoxes$) est retournée par l'opérateur.

Algorithm 2: `TEC`

```
   Input: N = (X, box, C) : an NCN, Φ : subcontractor,
          TECnodes : max calls, ε : precision parameter
   Output: box : the contracted box
1  Boxes ← {Φ(X, box, C, ε)}          /* queue (FIFO) of boxes */
2  AtomicBoxes ← ∅                    /* set of atomic boxes */
3  #nodes ← 1
4  while Boxes ≠ ∅ and #nodes ≤ TECnodes do
5      box' ← Boxes.pop()
6      if box' ≠ ∅ then
7          if ω(box') > ε then
8              (box1, box2) ← Bisect(box')
9              Boxes.push(Φ(X, box1, C, ε))
10             Boxes.push(Φ(X, box2, C, ε))
11             #nodes ← #nodes + 2
12         else
13             AtomicBoxes.push(box')
14 box ← Hull(Boxes ∪ AtomicBoxes)
```

3.2 Le choix de paramètres pour TEC

Nous détaillons dans cette section les paramètres choisis pour l'opérateur TEC. Ces choix ont été validés par des expérimentations préliminaires non reportées dans cet article.

Parcours en largeur d'abord

Le paramètre $TECnodes$ permet de borner le nombre de noeuds du sous-arbre TEC. L'utilisation d'un parcours en largeur d'abord découle naturellement de ce choix de paramètre. Soulignons toutefois qu'une première version de l'opérateur TEC bornait l'arbre TEC par sa hauteur, ce qui rendait indifférent l'ordre du parcours (en largeur ou en profondeur d'abord), l'arbre étant complet. En revanche, cette version avait un défaut majeur. Le doublement potentiel du nombre de noeuds à chaque niveau du sous-arbre rendait le contrôle de l'opérateur délicat. C'est pourquoi cette version de TEC bornée par la hauteur a été abandonnée au profit de la version présentée.

Stratégie de choix de variables

TEC ne rogne/bissecte pas l'ensemble des variables du réseau. Par conséquent, il est nécéssaire d'introduire une stratégie de choix de variables. Le choix

1. Notons que ce nombre est nécessairement impair dans l'implantation actuelle.

d'un sous-ensemble de variables prédéfinies a rapidement été écarté. En effet, un tel choix ne prend pas en compte l'évolution du réseau et peut manquer cruellement de pertinence dans certains cas. Notre choix s'est donc orienté vers une étude expérimentale des stratégies dynamiques de choix de variables.

Stratégie de branchement

Le branchement effectué en chaque nœud a aussi fait l'objet d'une étude expérimentale permettant d'observer le comportement de l'opérateur en fonction du nombre de sous-boîtes générées (bissection, trissection, ...). La bissection est la plus avantageuse au vu des résultats. L'utilisation de stratégies de choix de variables dynamiques semble expliquer ces résultats. En effet, on peut remarquer qu'une variable peut être traitée plusieurs fois dans une même branche de l'arbre. La stratégie peut alors sélectionner consécutivement deux fois la même variable, et ainsi appliquer un principe de trissection. L'avantage est que le choix du nombre de sections s'adapte ainsi dynamiquement en fonction de la contraction du domaine.

3.3 Valeurs par défaut des paramètres de TEC

Une étude expérimentale préliminaire a permis de calculer les valeurs par défaut de l'opérateur TEC. Ces valeurs permettent de définir une version de TEC stable et comparable à l'algorithme 3BCID [19] de l'état de l'art.

Le sous-contracteur Φ de TEC est un contracteur de programmation par contraintes (cf. la procédure CPcontract à l'algorithme 1). Des études non reportées dans cet article montrent que l'utilisation de HC4 est préférable à celle de 3BCID au sein de l'arbre TEC [16].

Selon l'instance considérée (cf. section 5), la meilleure valeur du paramètre $TECnodes$ oscille entre 10 et 100. Nos études ne nous ont pas permis de trouver un critère intelligent permettant de sélectionner $TECnodes$ selon l'instance traitée. Toutefois, la valeur 25 donne un compromis toujours acceptable et définit ainsi la valeur par défaut de $TECnodes$ lors de la comparaison à l'existant.

Enfin, nos comparaisons expérimentales des différentes stratégies de choix de variables existantes nous ont conduits à choisir la stratégie $SmearSumRel$ [2].

2. *SmearSumRel* est une stratégie de choix de variables connue comme l'une des plus performantes [18]. Elle se base sur la fonction *smear* [10]. $smear(x_i, c_j)$ mesure l'impact de la variable x_i sur la fonction de c_j. Son calcul implique la largeur de $[x_i]$ et la dérivée partielle de c_j par rapport à x_i évaluée sur la boîte $[X]$.

4 L'opérateur Graham-TEC

Nous présentons dans cette section une variante de TEC, nommée Graham-TEC, qui infère des contraintes binaires implicites. Ces contraintes linéaires sont ensuite ajoutées au polytope généré par l'opérateur de convexification polyhédrale utilisé pendant la résolution. Plus précisément, si l'on reprend l'algorithme 1, nous proposons de mettre en œuvre $CPContract$ par Graham-TEC, qui transmet des contraintes linéaires à la procédure suivante ($ConvexifyContract$) pour enrichir son polytope. Cet ajout compense la relaxation de l'espace solution par la convexification et permet ainsi de mieux contracter les domaines.

Comme le fait TEC, l'opérateur Graham-TEC construit un arbre en largeur d'abord. Il calcule ensuite, pour chaque paire de variables, l'enveloppe convexe des boîtes feuilles de l'arbre TEC. Finalement, l'algorithme retourne, comme TEC, une boîte contractée issue de l'enveloppe des boîtes feuilles, mais retourne en plus un ensemble de contraintes binaires linéaires formant un polyèdre convexe.

4.1 Construction du polytope

L'algorithme 3 est appelé à la fin de l'algorithme 2. Cet algorithme prend en entrée l'ensemble des boîtes feuilles de l'arbre TEC, la boîte $Hull$ de celles-ci ainsi qu'un paramètre γ permettant de décider si un demi-plan construit est ajouté ou non au polytope. Il retourne au final un ensemble de contraintes binaires linéaires.

Pour chaque paire (x_i, x_j) de variables, et pour chaque boîte feuille l de l'arbre TEC, les quatre sommets du rectangle correspondant à la projection de l sur les deux dimensions (x_i, x_j) sont ajoutés à une liste $Points$. Si un sommet est trop proche d'un sommet du rectangle $[h_i] \times [h_j]$ (correspondant à la projection de la boîte enveloppe h), le quadrant est considéré comme "couvert" et aucun demi-plan ne sera ajouté dans cette zone. Dans le cas extrême d'une seule boîte feuille confondue avec son enveloppe, ces conditions calculent bien une distance nulle pour chacun des 4 quadrants, si bien que `PolytopeGenerator` termine immédiatement sans construire de contraintes linéaires implicites. Les conditions précises décrites aux lignes 10 à 17 mesurent la distance à un sommet du rectangle $[h_i] \times [h_j]$ en pourcentage γ de la largeur $[h_i]$ ou $[h_j]$. La figure 1 montre un exemple.

Un algorithme de génération d'enveloppe convexe [9] est ensuite appelé sur chaque quadrant sélectionné. L'algorithme produit une liste ordonnée des points composant l'enveloppe convexe de tous les points de $Points$. Cet algorithme classique de géométrie algorithmique commence par trier les points candidats

Algorithm 3: `PolytopeGenerator`

```
   Input: h : the box representing the hull
   TECLeaves : set of TEC leaves, γ : the min gain
   Output: S : a set of linear constraints
1  S ← ∅
2  [h_i] ← h[x_i] ; [h_j] ← h[x_j]
3  foreach (x_i, x_j) ∈ X^2 with i < j do
4      Points ← ∅
5      foreach sector ∈ {NE, NW, SW, SE} do
6          cover(sector) ← false
7      foreach l ∈ TECLeaves do
8          [l_i] ← l[x_i] ; [l_j] ← l[x_j]
9          Points ← Points ∪ {(l̄_i, l̄_j), (l̄_i, l_j), (l_i, l̄_j), (l_i, l_j)}
10         if |h̄_i − l̄_i| < γ × ω(h_i) and |h_j − l_j| < γ × ω(h_j) then
11             cover(SE) ← true
12         if |h̄_i − l̄_i| < γ × ω(h_i) and |h̄_j − l̄_j| < γ × ω(h_j) then
13             cover(NE) ← true
14         if |h_i − l_i| < γ × ω(h_i) and |h̄_j − l̄_j| < γ × ω(h_j) then
15             cover(NW) ← true
16         if |h_i − l_i| < γ × ω(h_i) and |h_j − l_j| < γ × ω(h_j) then
17             cover(SW) ← true
18     foreach sector ∈ {NE, NW, SW, SE} do
19         if ¬cover(sector) then
20             //Graham outputs the pareto front on a sector
21             paretoFront ← Graham(Points, sector)
22             S ← S ∪ getRigorousConstraints(h, x_i,
23                                 x_j, paretoFront, sector)
24 return S
```

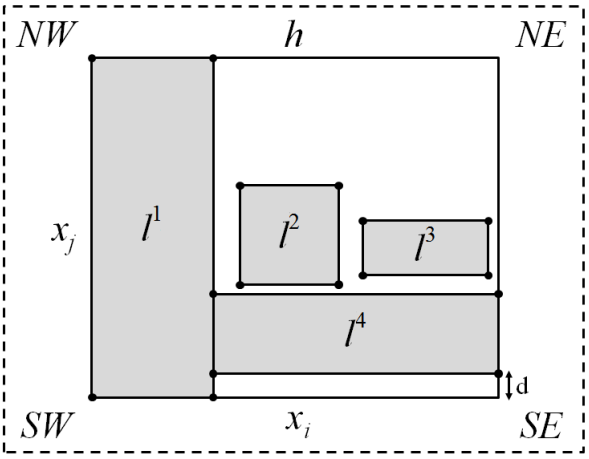


FIGURE 1 – Illustration de la procédure `PolytopeGenerator`. Le rectangle en pointillés représente la boîte initiale (avant contraction de l'opérateur) projetée sur x_i et x_j. h représente la boîte enveloppe retournée par l'opérateur. Les quadrants NW, SW et SE ne pouvant pas apporter de gain suffisant, ils ne sont pas traités (d étant inférieure à $\gamma \times \omega(h_j)$).

par angle polaire en fonction d'un point « pivot ». Il parcourt ensuite les points dans ce nouvel ordre en construisant l'enveloppe convexe dans le sens trigonométrique direct. L'algorithme décide de mettre le point courant dans l'enveloppe, plus précisément dans une liste *paretoFront*, si l'angle entre les deux derniers segments de l'enveloppe en construction est positif. La complexité en temps de cet algorithme est dominée par le tri en $O(p.log(p))$ où p est le nombre de points traités.

A partir de cette liste de points, la procédure `getRigorousConstraints` génère un ensemble de demi-plans (contraintes binaires linéaires) qui seront ajoutés au polytope créé par $ConvexifyContract$.

4.2 Génération rigoureuse des contraintes

La procédure `getRigorousConstraints` (cf. algorithme 4) permet de générer de manière rigoureuse des contraintes linéaires à partir des points de l'enveloppe convexe. Chaque paire de points consécutifs $((x_i, y_i), (x_{i+1}, y_{i+1}))$ prise dans la liste *paretoFront* est traitée itérativement. Le but est de générer une droite quasi-confondue avec le segment entre ces deux points. La difficulté réside dans le fait que la droite réelle $y = ax + b$ qui passe par ces deux points a généralement des coefficients a et b qui ne sont pas des nombres flottants (cf. figure 2). Or, les problèmes d'arrondis des calculs sur les flottants pourraient entraîner la génération d'un demi-plan à coefficients flottants qui n'englobe pas ces deux points. Pour éviter ce problème, les demi-plans sont calculés de manière rigoureuse grâce à l'arithmétique d'intervalles. A partir des points (x_i, y_i) et (x_{i+1}, y_{i+1}), l'arithmétique d'intervalles calcule des (tout petits) intervalles $[a]$ et $[b]$ contenant nécessairement les coefficients réels. Le lecteur pourra facilement vérifier le choix des bornes permettant d'approximer les deux coefficients de manière rigoureuse :

- pour les 4 quadrants, le coefficient directeur est donné par $\underline{a}$;
- pour les deux quadrants nord, la droite calculée est définie avec le coefficient $\overline{b}$;
- pour les deux quadrants sud, la droite calculée est définie avec le coefficient $\underline{b}$;

La figure 2 illustre ce calcul rigoureux sur le quadrant SE.

La ligne 19 ajoute une condition pour garantir que le point précédent satisfasse l'inégalité, c'est-à-dire que le demi-plan contienne aussi le point précédent (x_{i-1}, y_{i-1}). En effet, l'arrondi $\underline{a}$ peut entraîner la violation de cette contrainte en théorie (dans un cas pathologique où les 3 points seraient quasi-alignés).

Pour un quadrant donné, l'ensemble *paretoFront* peut contenir au maximum r points, avec r le nombre de boîtes feuilles du sous-arbre TEC. La boucle principale de l'algorithme 4 peut donc produire au plus $r-1$ contraintes linéaires. Certaines de ces contraintes linéaires peuvent être presque parallèles entre elles (ou parallèles aux bords de la boîte). Ces contraintes ne couperaient alors presque pas de volume de la boîte et ralentiraient l'opérateur $ConvexifyContract$ sans en

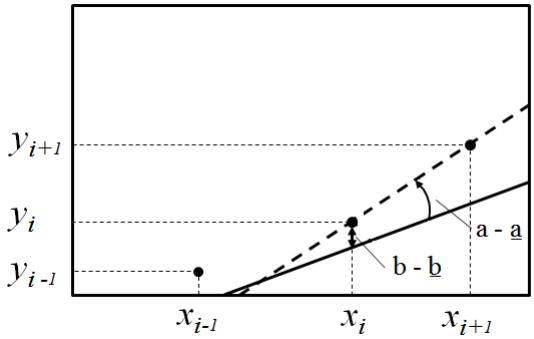


FIGURE 2 – Schéma représentant le calcul rigoureux sur le quadrant SE. La droite en pointillés correspond à la droite de coefficients a et b à valeur dans $\mathbb{R}$. La deuxième correspond à la droite de coefficients a et b arrondis sur les nombres flottants.

Algorithm 4: `getRigorousConstraints`

Input: $paretoFront = ((x_1, y_1), \ldots, (x_m, y_m))$, h the box representing the hull, x and y two dimensions, $sector$
Output: P a set of rigorous half-planes

$P \leftarrow \emptyset$
foreach $(x_i, y_i) \in paretoFront$ **do**
  //Build the degenerated intervals :
  $[x_i] \leftarrow [x_i, x_i]$; $[y_i] \leftarrow [y_i, y_i]$
  $[x_{i+1}] \leftarrow [x_{i+1}, x_{i+1}]$; $[y_{i+1}] \leftarrow [y_{i+1}, y_{i+1}]$
  //Interval of slope value of the linear function
  $[a] \leftarrow ([y_{i+1}] - [y_1])/([x_{i+1}] - [x_i])$
  //Interval of y-intercept value of the linear function
  $[b] \leftarrow [y_i] - ([x_i] \times [a])$
  switch $sector$ **do**
    case SE
      $p \leftarrow \underline{a} \times x - y \leq -\underline{b}$
    case NE
      $p \leftarrow \underline{a} \times x - y \geq -\overline{b}$
    case NW
      $p \leftarrow \underline{a} \times x - y \geq -\overline{b}$
    case SW
      $p \leftarrow \underline{a} \times x - y \leq -\underline{b}$
  if (x_{i-1}, y_{i-1}) satisfies p **then** $P \leftarrow P \cup \{p\}$
return $P' \subseteq P$

améliorer la contraction. Aussi, à la dernière ligne de *getRigorousConstraints*, nous prévoyons de ne renvoyer qu'un sous-ensemble des contraintes générées. On pourrait imaginer de ne retourner qu'un top-k des contraintes rognant le plus d'espace, ce qui reviendrait à explorer $\binom{k}{r-1}$ combinaisons. Le choix des contraintes à conserver implanté dans la version courante est assez simple. Nous utilisons un algorithme glouton qui traite chaque contrainte linéaire individuellement et vérifie qu'elle rogne au moins une proportion γ sur l'une des dimensions. Ce test est illustré sur la figure 3. La contrainte en pointillés rouges ne permet de rogner qu'une portion d sur la dimension x_i, inférieure à $\gamma \times h[x_i]$, et elle n'est donc pas posée. En revanche, la contrainte linéaire en noir permet de couper significativement $[h_i]$ et $[h_j]$ et sera donc transmise à *ConvexifyContract*. Cette procédure glouton ne filtre en quelque sorte que les contraintes linéaires presque parallèles aux axes, modulo une normalisation des deux largeurs $\omega([h_i])$ et $\omega([h_i])$. Nous l'étendrons à court terme pour prendre en compte les contraintes parallèles entre elles. Elle permet de réduire en pratique le nombre de contraintes transmises à *ConvexifyContract*, réduction que nous quantifierons expérimentalement.

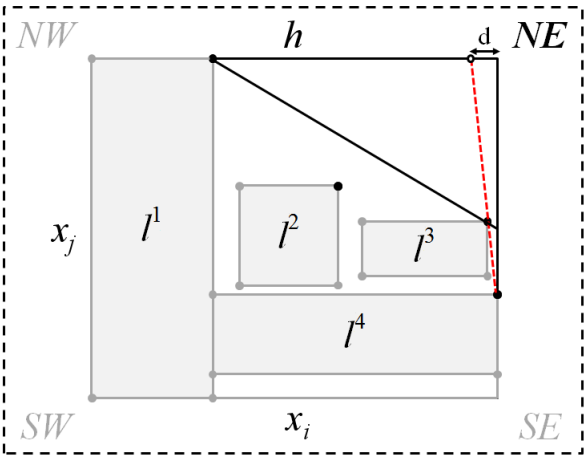


FIGURE 3 – Illustration de la procédure `getRigorousConstraints`. La droite en pointillés n'apportant pas un gain suffisant par rapport au bord de la boîte, elle n'est pas ajoutée au polytope généré.

5 Validation expérimentale

Dans cette section, nous présentons les résultats de l'opérateur TEC et de son extension Graham-TEC, et nous les comparons aux deux opérateurs de contraction HC4 et 3BCID définissant l'état de l'art.

5.1 Bancs d'essais et outil de résolution

Nous avons implanté TEC et Graham-TEC dans la bibliothèque à intervalles libre `Ibex` (Interval-Based EXplorer) [7]. Cette bibliothèque a facilité l'implantation de nos opérateurs en fournissant un certain nombre de briques telles que la stratégie de résolution *Branch & Contract*décrite par l'algorithme 1, HC4 et 3BCID comme contracteurs issus de la PPC (*CPContract*) et X-Newton comme opérateur de convexification polyhédrale (*ConvexifyContract*).

Les comparaisons sont faites sur des problèmes de résolution traités par une stratégie proche de celle décrite par l'algorithme 1, mais aussi sur des problèmes d'optimisation globale sous contraintes traités par l'optimiseur IbexOpt [18]. En plus du branchement et de la contraction (qui enchaîne *CPContract* et *ConvexifyContract*), IbexOpt cherche à calculer en chaque nœud de l'arbre un minorant et un majorant de la fonction objectif dont la différence décide de

la terminaison. *CPContract* (cf. algorithme 1, ligne 5) représente les opérateurs que nous comparons (HC4, 3BCID, TEC, Graham-TEC) tandis que l'opérateur X-Newton (*ConvexifyContract*) est utilisé dans chacune des stratégies. L'heuristique de choix de variables utilisée est l'heuristique *SmearSumRel* présentée à la section 2.

Tous les paramètres ont été fixés à des valeurs communes à toutes les instances testées. La précision d'une boîte solution (cf. algorithme 1, ligne 8) a été fixée à ϵ = `1.e-10` et le temps-limite de chaque instance a été fixé à 1800 secondes. Pour les instances d'optimisation, la valeur du paramètre de précision sur la fonction objectif a été fixée à ϵ_{obj} = `1.e-08`. Les paramètres de TEC ainsi que ceux de Graham-TEC utilisés lors de nos expériences sont ceux présentés à la section 3.3, à savoir : $TECnodes$=25, bissection=SmearSumRel. L'ensemble de nos expériences ont été réalisées sur un *Intel* $64\,bits$ $2.6\,GHz$, doté de $32\,Go$ de RAM sous *Linux*.

Nous avons effectué nos différentes expériences sur deux bancs d'essais. Un échantillon de 23 instances provenant du banc d'essais COPRIN [3], qui porte sur des problèmes de résolution et un échantillon de 82 instances provenant du banc d'essais COCONUT [4] qui porte sur des problèmes d'optimisation globale sous contraintes numériques. Ces instances ont fait l'objet d'une sélection lors de travaux de membres de l'équipe Ibex portant sur un autre sujet [14]. Elles correspondent à toutes les instances difficiles des séries 1 et 2, mais ne dépassant pas 50 variables [5].

5.2 Résultats obtenus par TEC

Sur les 23 instances en résolution, TEC est légèrement meilleur que HC4 en temps de résolution (-7% en moyenne) mais a une capacité de contraction plus importante (-53% du nombre de nœuds explorés en moyenne). Notons que l'opérateur X-Newton utilisé après l'opérateur HC4 permet le plus gros de la contraction. En effet, en laissant HC4 comme seul contracteur de la stratégie, le temps de résolution explose et dépasse souvent le temps-limite. En revanche, l'opérateur 3BCID permet une meilleure contraction que TEC sur certaines instances, avec une perte d'un facteur 2 en temps de résolution sur 3 instances (`butcher8-a`, `butcher8-b` et `Synthesis`).

Nous présentons plus en détail les résultats obtenus sur les 82 instances d'optimisation globale sous contraintes. La figure 4 présente le comparatif des opérateurs HC4 et TEC en fonction de leurs temps d'exécution.

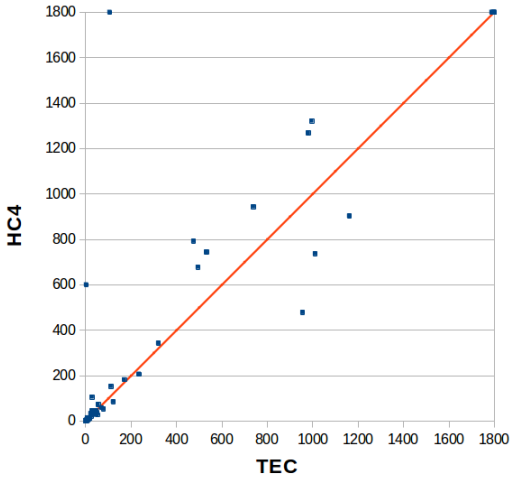


FIGURE 4 – TEC versus HC4 : Les coordonnées de chaque point représente le temps d'exécution (en secondes) requis par les deux stratégies.

Sur 2 instances, TEC obtient un gain en temps supérieur à un facteur 2 par rapport à HC4. En particulier, l'instance `haldmads` est résolue en moins de 120 secondes par TEC alors qu'elle n'est pas résolue par HC4 dans le temps-limite de 30 mn. Le tableau 1 présente les gains détaillés en temps d'exécution de TEC par rapport à HC4.

	loss $< .2$	loss $[.2,.5]$	loss $[.5,.7]$	loss $[.7,.9]$	equiv $[.9,1.1]$	gain $[1.1,1.4]$	gain $[1.4,2]$	gain $[2,5]$	gain >5
$\leq$ 60s	0	2	6	12	13	10	4	2	1
$>$ 60s	0	0	1	4	17	6	2	0	2

TABLE 1 – Gains détaillés de TEC par rapport à HC4. Le gain est exprimé par $\frac{temps(HC4)}{temps(TEC)}$. Les valeurs représentent le nombre d'instances (résolues en moins de 60s et en plus de 60s) correspondant à chaque plage de gains.

Bien que TEC permette d'améliorer significativement le temps d'exécution sur certaines instances difficiles, il est souvent non compétitif avec HC4 sur des instances faciles (rapidement résolues). Une explication à ce phénomène tient au surcoût de la contraction supplémentaire. Le tableau 2 permet de visualiser les gains détaillés en nombre de nœuds explorés par la stratégie utilisant TEC par rapport à celle utilisant HC4. Ces résultats confirment le gain en contraction.

	loss $< .2$	loss $[.2,.5]$	loss $[.5,.7]$	loss $[.7,.9]$	equiv $[.9,1.1]$	gain $[1.1,1.4]$	gain $[1.4,2]$	gain $[2,5]$	gain >5
$\leq$ 60s	0	0	0	1	27	13	4	3	2
$>$ 60s	0	0	0	0	11	12	2	4	3

TABLE 2 – TEC vs. HC4 : Résumé des gains en *#noeuds*

La figure 5 présente le comparatif des opérateurs 3BCID et TEC en fonction de leurs temps d'exécution.

3. `www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html`

4. `www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`

5. Peu d'instances dépassant cette taille peuvent être traitées aujourd'hui par les optimiseurs déterministes (à intervalles ou non rigoureux d'ailleurs) sans paramétrage ad-hoc

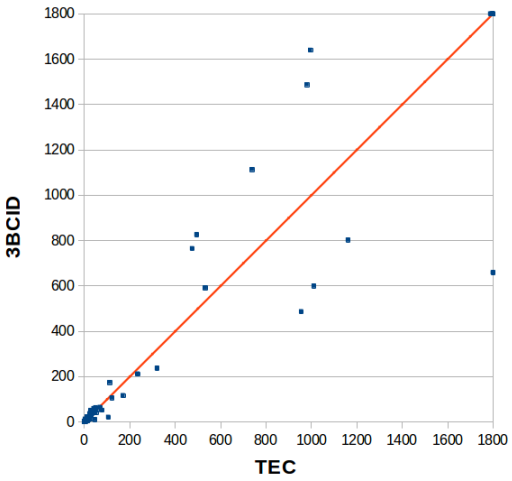


FIGURE 5 – TEC vs. 3BCID (temps CPU)

TEC et 3BCID sont deux opérateurs de contraction forte. Par conséquent, ces deux opérateurs sont coûteux en chaque nœud. Sur la figure 5 on constate que sur certaines instances difficiles, TEC semble plus approprié que 3BCID et sur certaines autres 3BCID obtient de meilleurs résultats. Toutefois, 3BCID permet de résoudre l'instance `ex8_4_5` tandis que TEC ne le permet pas, même en étendant le temps-limite à une heure. Le tableau 3, présente les gains détaillés en temps d'exécution.

	loss < .2	loss [.2,.5]	loss [.5,.7]	loss [.7,.9]	equiv [.9,1.1]	gain [1.1,1.4]	gain [1.4,2]	gain [2,5]	gain >5
≤ 60s	1	1	6	12	13	10	5	2	0
> 60s	1	0	4	2	17	2	6	0	0

TABLE 3 – TEC vs. 3BCID : Résumé des gains en temps

Ce tableau montre une répartition quasiment uniforme des instances résolues en plus de 60 s. En effet, 3BCID est plus performant sur 7 instances et TEC sur 8 instances pour les instances résolues en plus de 60 s. Pour celles résolues en moins d'une minute, 3BCID est légèrement meilleur. Cette observation nous a poussés à explorer des versions hybrides de ces deux opérateurs, toutefois, sans résultats.

Le tableau 4 permet de visualiser les gains détaillés en nombre de nœuds de l'arbre d'exploration de TEC par rapport à 3BCID.

	loss < .2	loss [.2,.5]	loss [.5,.7]	loss [.7,.9]	equiv [.9,1.1]	gain [1.1,1.4]	gain [1.4,2]	gain [2,5]	gain >5
≤ 60s	1	1	3	9	24	9	2	1	0
> 60s	1	0	2	7	15	6	0	1	0

TABLE 4 – TEC vs. 3BCID : Résumé des gains en *#noeuds*

La répartition des instances en fonction du gain en nombre de nœuds de l'arbre d'exploration montre que l'opérateur 3BCID semble apporter plus fréquemment une contraction supérieure. Toutefois, cette différence reste mineure.

5.3 Résultats obtenus par Graham-TEC

L'opérateur Graham-TEC obtient de meilleurs résultats que l'opérateur TEC pour les problèmes de résolution. Graham-TEC améliore les temps d'exécution sur les 23 instances par rapport à HC4 (−12% en moyenne) et produit une contraction supérieure, avec une réduction moyenne de 63% du nombre de nœuds explorés. Toutefois, Graham-TEC reste moins performant que 3BCID sur ce benchmark (+5% en moyenne en temps d'exécution).

Sur les instances d'optimisation, Graham-TEC améliore aussi l'opérateur TEC. La figure 6 présente le comparatif des opérateurs HC4 et Graham-TEC en fonction de leurs temps d'exécution.

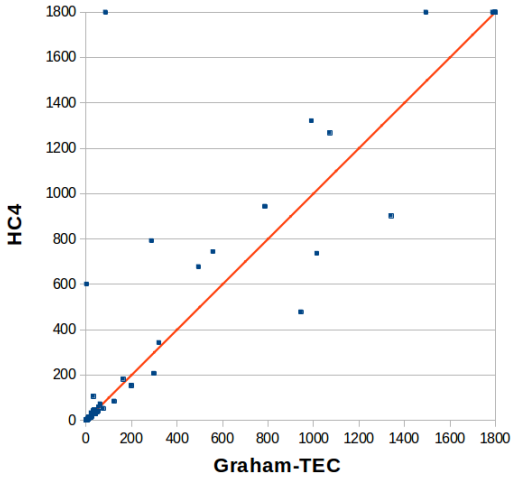


FIGURE 6 – Graham-TEC vs. HC4 : (temps CPU)

Pour les instances résolues en plus de 400 secondes, Graham-TEC est plus performant sur 9 instances alors que HC4 l'est sur 3 instances. Parmi les 9 instances où Graham-TEC est plus performant, 2 d'entres elles ne sont pas résolues par HC4 (`haldmads` et `disc2`). De plus, même en étendant la limite de temps à 3600 s, HC4 ne permet pas de résoudre ces deux instances.

Le tableau 5, présentant les gains détaillés en temps d'exécution montre que HC4 est majoritairement plus performant sur les instances résolues en moins de 60 s.

	loss < .2	loss [.2,.5]	loss [.5,.7]	loss [.7,.9]	equiv [.9,1.1]	gain [1.1,1.4]	gain [1.4,2]	gain [2,5]	gain >5
≤ 60s	0	2	8	13	13	9	2	2	1
> 60s	0	0	3	3	16	7	0	1	3

TABLE 5 – Graham-TEC vs. HC4 : Résumé des gains en temps

De manière identique à TEC, Graham-TEC est un opérateur de contraction forte qui peut entraîner un surcoût sur des instances faciles.

La figure 7 présente le comparatif des opérateurs 3BCID et Graham-TEC en fonction de leurs temps d'exécution.

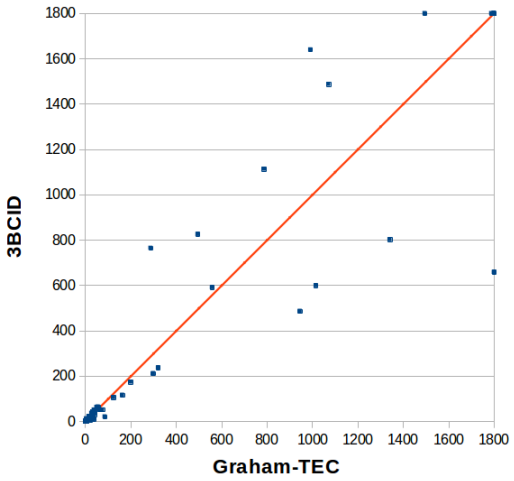


FIGURE 7 – Graham-TEC vs. 3BCID : (temps CPU)

Pour les instances résolues en plus de 400 secondes, Graham-TEC est plus performant que 3BCID sur 7 instances dont une instance (`disc2`) n'est pas résolue par 3BCID, même en étendant le temps-limite à une heure. A l'inverse, l'instance `ex8_4_5`, résolue uniquement par 3BCID, n'est pas résolue par Graham-TEC avec un temps-limite étendu à 3600 s. Le tableau 6 présente les gains détaillés en temps d'exécution.

	loss < .2	loss [.2,.5]	loss [.5,.7]	loss [.7,.9]	equiv [.9,1.1]	gain [1.1,1.4]	gain [1.4,2]	gain [2,5]	gain >5
≤ 60s	1	1	5	10	19	8	4	2	0
> 60s	1	0	4	4	15	4	2	1	1

TABLE 6 – Graham-TEC vs. 3BCID : Résumé des gains en temps

Pour les instances résolues en plus de 60 s, Graham-TEC obtient un gain d'un facteur supérieur à 2 sur 2 instances tandis que 3BCID n'obtient un gain d'un facteur supérieur à 2 que sur une seule instance. Toutefois, la répartion des instances reste quasiment uniforme. Le tableau 7 présentant le gain en nombre de nœuds de l'arbre d'exploration montre que Graham-TEC permet une contraction plus importante que 3BCID.

	loss < .2	loss [.2,.5]	loss [.5,.7]	loss [.7,.9]	equiv [.9,1.1]	gain [1.1,1.4]	gain [1.4,2]	gain [2,5]	gain >5
≤ 60s	1	1	1	6	23	12	4	1	1
> 60s	1	0	2	5	8	12	2	1	1

TABLE 7 – Graham-TEC vs. 3BCID : Résumé des gains en *#noeuds*

Ces résultats s'expliquent par le fait que Graham-TEC reste encore coûteux sur certaines instances à cause du travail supplémentaire effectué sur les contraintes inférées. Malgré nos améliorations, ce surcoût est parfois contreproductif, en particulier sur les instances contenant un grand nombre de variables. De ce fait, malgré une contraction potentiellement plus forte, le surcoût occasionné par les contraintes inférées peut, dans certains cas, engendrer une perte de performance à cause d'un faible gain en contraction.

Une étude qualitative, sur l'ensemble des 82 instances étudiées, indique qu'en moyenne Graham-TEC infère 12 contraintes par nœud. On peut constater une augmentation drastique de cette moyenne en fonction du nombre de variables. En effet, sur les instances de moins de 9 variables, la moyenne est de 6 contraintes, pour celles entre 9 et 14 variables, la moyenne est de 8 contraintes tandis que pour les instances de plus de 14 variables, la moyenne grimpe à 35 contraintes par nœud. Toutefois, il est important de remarquer que la perte de performance de Graham-TEC n'est pas directement liée à l'augmentation du nombre de contraintes inférées. Le gain de contraction peut compenser le surcoût engendré, apportant alors un gain de performance même sur des instances avec une moyenne élevée.

6 Travaux existants

Le principe de convexification polyédrale de l'ensemble-solution est souvent utilisé en optimisation et en programmation mathématique [17]. Il consiste à calculer une approximation extérieure de l'espace-solution en convexifiant chaque contrainte d'inégalité individuellement. Une approximation polyédrale de l'espace-solution permet de réduire l'espace de recherche et/ou d'améliorer la borne inférieure de la fonction objective à minimiser en faisant appel à un résolveur de programmation linéaire. L'idée de TEC est significativement différente puisque les contraintes linéaires apprises dépendent du nœud courant. Elles ne correspondent pas aux contraintes initiales mais sont déduites des réductions obtenues durant l'exploration de l'arbre TEC.

L'approche de Pelleau et al. [15] s'inspire de travaux effectués en interprétation abstraite pour étendre la notion de boîte dans le CSP numérique. Au lieu de représenter les domaines par des boîtes, chaque paire (i, j) de variables du système est associée à un *octogone* qui peut se voir comme l'intersection de deux boîtes, la boîte « initiale » et une boîte ayant subie une rotation de 45 degrés. Pour chaque chaque paire (i, j), quatre contraintes $\pm x_i \pm x_j \leq c_k$ sont ajoutées. Exprimer les contraintes dans la boîte pivotée revient à remplacer dans leur expression les variables x_i et x_j par leur représentation dans la nouvelle base. Gérer ces domaines étendus revient ainsi à contracter un système avec de nombreuses contraintes additionnelles.

Un contracteur arborescent proche de TEC a été introduit en 2009 [1]. Des sous-systèmes carrés de contraintes d'égalité (avec autant d'équations que d'inconnues et donc un ensemble fini de solution) doivent être préalablement sélectionnés dans le réseau. A chaque noeud de l'arbre de recherche général, une contraction basée sur une exploration arborescente est

lancée pour chaque sous-système. A la différence de l'arbre TEC, un appel au sous-contracteur (HC4) est suivi par un appel à Newton sur intervalles qui exploite le fait que le sous-système est carré, ce qui permet de converger parfois plus vite vers les solutions. Ce contracteur, appelé Box-k, obtient de bons résultats sur des systèmes de contraintes peu denses. Sa limite actuelle est de demander à l'utilisateur d'identifier un ensemble de sous-systèmes qui seront traités par Box-k. Au final, TEC est plus généraliste que Box-k et s'applique sur tout le système (quoique peu de variables soient en pratique bissectées à chaque appel compte-tenu de la faible valeur de TECnodes).

7 Conclusion

Cet article a introduit deux nouveaux opérateurs de contraction arborescents obtenant de bonnes performances par rapport à l'état de l'art. L'opérateur Graham-TEC permet de faire un lien entre les deux principaux types d'opérateurs de contraction utilisés dans la plupart des résolveurs à intervalles, à savoir les opérateurs de contraction issus de la programmation par contraintes et ceux reposant sur une convexification polyhédrale. Nos premiers résultats expérimentaux montrent l'intérêt de cette approche qui obtient des gains significatifs par rapport à l'état de l'art sur des instances difficiles d'optimisation globale.

L'opérateur Graham-TEC reste en revanche trop coûteux sur certains types d'instances, en particulier lorsque l'on monte en dimension. En effet, cet opérateur traite toutes les paires de variables et peut ainsi générer un nombre quadratique de contraintes linéaires. Bien que des heuristiques de sélection de contraintes aient été élaborées, il faut encore améliorer cet opérateur pour mieux prendre en compte cette croissance.

Références

[1] I. Araya, B. Neveu, and G. Trombettoni. A New Monotonicity-Based Interval Extension Using Occurrence Grouping. In *Workshop IntCP*, 2009.

[2] I. Araya, G. Trombettoni, and B. Neveu. Exploiting Monotonicity in Interval Constraint Propagation. In *Proc. AAAI*, pages 9–14, 2010.

[3] I. Araya, G. Trombettoni, and B. Neveu. A Contractor Based on Convex Interval Taylor. In *CPAIOR 2012*, number 7298 in LNCS, pages 1–16, May 2012.

[4] A. Balafrej, C. Bessiere, E.H. Bouyakhf, and G. Trombettoni. Adaptive Singleton-based Consistencies. In *AAAI*, pages 2601–2607. AAAI Press, 2014.

[5] F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, pages 230–244, 1999.

[6] H. Bennaceur and M.-S. Affane. Partition-k-AC : An Efficient Filtering Technique Combining Domain Partition and Arc Consistency. In *Proc. CP*, pages 560–564, 2001.

[7] G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173 :1079–1100, 2009.

[8] R. Debruyne and C. Bessiere. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proc. IJCAI*, pages 412–417, 1997.

[9] R. Graham. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Inf. Process. Lett.*, 1(4) :132–133, 1972.

[10] R.B. Kearfott and M. Novoa III. INTBIS, a portable interval Newton/Bisection package. *ACM Trans. on Mathematical Software*, 16(2) :152–157, 1990.

[11] O. Lhomme. Consistency Techniques for Numeric CSPs. In *IJCAI*, pages 232–238, 1993.

[12] F. Messine. *Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution des problèmes avec contraintes.* PhD thesis, LIMA-IRIT-ENSEEIHT-INPT, Toulouse, 1997.

[13] A. Neumaier. *Interval Methods for Systems of Equations.* Cambridge Univ. Press, 1990.

[14] B. Neveu, G. Trombettoni, and I. Araya. Node Selection Strategies in Interval Branch and Bound Algorithms. *under submission*, page , 2015.

[15] M. Pelleau, C. Truchet, and F. Benhamou. Octagonal Domains for Continuous Constraints. In *Proc. CP, Constraint Programming, LNCS 6876*, pages 706–720. Springer, 2011.

[16] O. Sans. Opérateur arborescent de filtrage pour le CSP numérique. Stage de master, University of Montpellier, 2014.

[17] M. Tawarmalani and N. V. Sahinidis. A Polyhedral Branch-and-Cut Approach to Global Optimization. *Mathematical Programming*, 103(2) :225–249, 2005.

[18] G. Trombettoni, I. Araya, B. Neveu, and G. Chabert. Inner Regions and Interval Linearizations for Global Optimization. In *AAAI*, pages 99–104, 2011.

[19] G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *Proc. CP, LNCS 4741*, pages 635–650, 2007.

Calcul du cube de skypatterns

Willy Ugarte[1] **Patrice Boizumault**[1] **Samir Loudni**[1] **Bruno Crémilleux**[1]

[1] GREYC (CNRS UMR 6072),
Université de Caen Basse-Normandie, 14032 CAEN
{prénom.nom}@unicaen.fr

Abstract

In [5], we introduce skypattern cubes and propose an efficient bottom-up approach to compute them. Our approach relies on derivation rules collecting skypatterns of a parent node from its child nodes without any dominance test. Non-derivable skypatterns are computed on the fly thanks to Dynamic CSP. The bottom-up principle enables to provide a concise representation of the cube based on skypattern equivalence classes without any supplementary effort. Experiments on mutagenicity datasets show the effectiveness of our proposal.

1. Introduction

La notion de requêtes *skyline* [1] a été récemment intégrée dans la découverte de motifs pour extraire des motifs appelés *skypatterns* [4, 6]. Les skypatterns sont des motifs basés sur la notion de Pareto-dominance pour lesquels aucune mesure ne peut être améliorée sans en dégrader au moins une autre. De tels motifs sont intéressants car ils n'obligent pas à fixer de seuil sur les mesures et possèdent un très fort intérêt global.

Dans la pratique, l'utilisateur ne connait pas a priori le rôle exact de chaque mesure, et ne peut déterminer à l'avance le sous-ensemble le plus approprié de mesures. De façon similaire au cube de skylines [3], l'utilisateur aimerait disposer du *cube de skypatterns*. Chaque élément du cube est un nœud qui associe, à un sous-ensemble des mesures, son ensemble de skypatterns. De plus, l'utilisateur peut facilement repérer les sous-ensembles de mesures ayant le même ensemble de skypatterns (qui forment une classe d'équivalence).

2. Contexte et définitions

Soit $\mathcal{I}$ un ensemble de littéraux appelés *items*. Un motif est un sous-ensemble non-vide de $\mathcal{I}$. Le langage d'itemsets correspond à $\mathcal{L}_\mathcal{I} = 2^\mathcal{I} \backslash \emptyset$. Un jeu de données est un multiset de motifs appelées transactions. La Figure 1a représente un jeu de données $\boldsymbol{r}$ où chaque transaction t_i est décrite par les items notés $A, \ldots, F$.

Exemple 1. Pour le jeu de données de Fig. 1a, on a $\texttt{freq}(BC)$=5, $\texttt{area}(BC)$=10 et $\texttt{mean}(BCD.price)$=25, avec les mesures suivantes définies par :

- $\texttt{freq}(\texttt{x}) = |\{t \in \boldsymbol{r} \mid \texttt{x} \subseteq t\}|$.
- $\texttt{area}(\texttt{x}) = \texttt{freq}(\texttt{x}) \times \texttt{taille}(\texttt{x})$ où $\texttt{taille}(\texttt{x})=|\texttt{x}|$.
- $\texttt{min}(\texttt{x}.att)$ (resp. $\texttt{max}(\texttt{x}.att)$) est la plus petite (resp. grande) valeur de x pour l'attribut att.
- $\texttt{mean}(\texttt{x}) = (\texttt{min}(\texttt{x}.att) + \texttt{max}(\texttt{x}.att))/2$.

Les skypatterns permettent d'exprimer une préférence de l'utilisateur via une relation de dominance [4].

Définition 1 (Dominance Pareto). Soit M un ensemble de mesures, un motif $\texttt{x}_i$ domine un autre motif $\texttt{x}_j$ sur M (noté $\texttt{x}_i \succ_M \texttt{x}_j$), ssi $\forall m \in M, m(\texttt{x}_i) \geq m(\texttt{x}_j)$ et $\exists m \in M, m(\texttt{x}_i) > m(\texttt{x}_j)$.

Définition 2 (Skypattern et opérateur skypattern). Soit M un ensemble de mesures, un skypattern sur M est un motif non-dominé. L'opérateur skypattern est $Sky(M) = \{\texttt{x}_i \in \mathcal{L}_\mathcal{I} \mid \nexists \texttt{x}_j \in \mathcal{L}_\mathcal{I}, \texttt{x}_j \succ_M \texttt{x}_i\}$

Exemple 2. Pour $M = \{\texttt{freq}, \texttt{area}\}$, on a $Sky(M) = \{BCDE, BCD, B, E\}$ (cf Figure 1b).

Soit M un ensemble de mesures, deux motifs $\texttt{x}_i$ et $\texttt{x}_j$ sont *indistincts* sur M (noté $\texttt{x}_i =_M \texttt{x}_j$) ssi $\forall m \in M$, $m(\texttt{x}_i)=m(\texttt{x}_j)$. $\texttt{x}_i$ et $\texttt{x}_j$ sont *incomparables* sur M (noté $\texttt{x}_i \prec\succ_M \texttt{x}_j$) ssi $(\texttt{x}_i \not\succ_M \texttt{x}_j)$ et $(\texttt{x}_j \not\succ_M \texttt{x}_i)$ et $(\texttt{x}_i \neq_M \texttt{x}_j)$.

Définition 3 (Skypattern incomparable). Un motif $\texttt{x} \in Sky(M)$ est incomparable sur M ssi $\forall \texttt{x}_i \in Sky(M)$ tel que $\texttt{x}_i \neq \texttt{x}, \texttt{x}_i \prec\succ_M \texttt{x}$.

Définition 4 (Skypattern indistinct). Un motif $\texttt{x} \in Sky(M)$ est indistinct sur M ssi $\exists \texttt{x}_i \in Sky(M)$ tel que $(\texttt{x}_i \neq \texttt{x}) \wedge (\texttt{x}_i =_M \texttt{x})$.

On peut regrouper les skypatterns indistincts.

Définition 5 (Groupe de skypatterns indistincts (GSI)). $S \subseteq Sky(M)$ est un GSI ssi $|S| \geq 2$ et $\forall \texttt{x}_i, \texttt{x}_j \in S, (\texttt{x}_i =_M \texttt{x}_j) \wedge \forall \texttt{x}_i \in S, \forall \texttt{x}_j \in Sky(M) \backslash S, (\texttt{x}_i \prec\succ_M \texttt{x}_j)$.

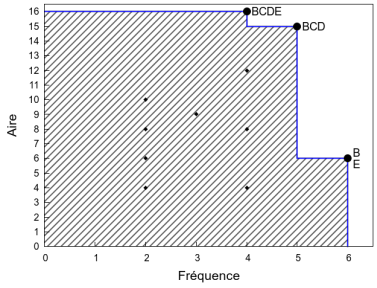
Exemple 3. Pour $M = \{\texttt{freq}, \texttt{area}\}$, $BCDE$ et BCD sont incomparables. B et E (indistincts) forment un GSI.

Définition 6 (Cube de skypatterns pour M). $SkyCube(M) = \{(M_u, Sky(M_u)) \mid M_u \subseteq M, M_u \neq \emptyset\}$

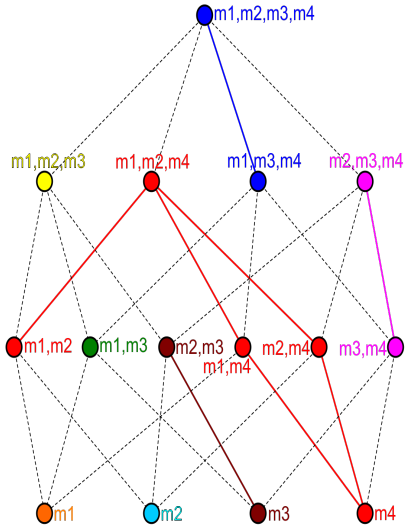
Trans.	Items					
t_1		B			E	F
t_2		B	C	D		
t_3	A				E	F
t_4	A	B	C	D	E	
t_5		B	C	D	E	
t_6		B	C	D	E	F
t_7	A	B	C	D	E	F

Item	A	B	C	D	E	F
Prix	30	40	10	40	70	55

(a) Jeu de données $\boldsymbol{r}$.



(b) Skypatterns pour M={freq, area}.



(c) Treillis associé à M.

Sous-ensemble de M	Ensemble de skypatterns
$\{m_1, m_2, m_3, m_4\}$	{BCDE, BCD, BDE, EF, BE, E}
$\{m_1, m_2, m_3\}$	{BCDE, BCD, BE, E}
$\{m_1, m_2, m_4\}$	{E}
$\{m_1, m_3, m_4\}$	{BCDE, BCD, BDE, EF, BE, E}
$\{m_2, m_3, m_4\}$	{BCDE, BDE, EF, E}
$\{m_1, m_2\}$	{E}
$\{m_1, m_3\}$	{BCDE, BCD, B, E}
$\{m_1, m_4\}$	{E}
$\{m_2, m_3\}$	{BCDE}
$\{m_2, m_4\}$	{E}
$\{m_3, m_4\}$	{BCDE, BDE, EF, E}
$\{m_1\}$	{B, E}
$\{m_2\}$	{ABCDEF, ABCEF, ABDEF, ABEF, ABCDE, ABCE, ABDE, ABE, ACDEF, ACEF, ACDE, ACE, ADEF, ADE, AEF, AE, BCDEF, BCEF, CDEF, CEF, BCDE, BCE, CDE, CE, BDEF, DEF, BDE, DE, BEF, EF, BE, E}
$\{m_3\}$	{BCDE}
$\{m_4\}$	{E}

(d) Cube de skypatterns pour M.

FIGURE 1 – $M = \{m_1 : \texttt{freq}, m_2 : \texttt{max}, m_3 : \texttt{area}, m_4 : \texttt{mean}\}$.

Exemple 4. La Figure 1c représente le treillis associé à M. La Figure 1d associe à chaque sous-ensemble non-vide de M son ensemble de skypatterns.

3. Règles de dérivation et calcul du cube

Deux règles de dérivation permettent construire de manière ascendante le cube de skypatterns, : l'une pour les skypatterns incomparables (cf. le théorème 1) et l'autre pour les GSI (cf. le théorème 2).

Théorème 1 (*Règle pour les incomparables*). Soit $M_u \subseteq M$, si $\mathtt{x}$ est un skypattern incomparable pour M_u, alors $\forall m \in M \setminus M_u$, $\mathtt{x} \in Sky(M_u \cup \{m\})$.

Théorème 2 (*Règle pour les GSI*). Soient $M_u \subseteq M$ et S un GSI pour M_u. $\forall m \in M \setminus M_u$, $\forall \mathtt{x} \in S$ t.q. $m(\mathtt{x}) = \max_{\mathtt{x}' \in S} \{m(\mathtt{x}')\}$, $\mathtt{x} \in Sky(M_u \cup \{m\})$.

Mais, ces deux règles ne permettent pas toujours de déterminer tous les skypatterns d'un nœud père à l'aide des skypatterns de ses fils.

Exemple 5. Pour M_u={m_1, m_3}, les skypatterns dérivés sont : B, E et $BCDE$ (le motif $BCDE$ est incomparable alors que les motifs B et E sont indistincts). Mais, les deux règles ne permettent pas de déduire que $BCD \in Sky(M_u)$.

Les CSP Dynamiques permettent de calculer à la volée les skypatterns manquants (non-dérivables). Considérons la séquence $P_1, \ldots, P_n$ de CSP où chaque $P_i = (\{\mathtt{x}\}, \mathcal{L}_\mathcal{I}, \mathtt{q}_i(\mathtt{x}))$ et :
- $\mathtt{q}_1(\mathtt{x}) = (B \not\succ_{M_u} \mathtt{x}) \wedge (E \not\succ_{M_u} \mathtt{x}) \wedge (BCDE \not\succ_{M_u} \mathtt{x})$.
- $\mathtt{q}_{i+1}(\mathtt{x}) = \mathtt{q}_i(\mathtt{x}) \wedge (s_i \not\succ_{M_u} \mathtt{x})$ où s_i est la première solution à la requête $\mathtt{q}_i(\mathtt{x})$.

Chaque requête $\mathtt{q}_i(\mathtt{x})$ permet d'agrandir la zone de non-dominance jusqu'à ce qu'elle soit totalement déterminée, i.e. $\exists n$ t.q. $\mathtt{q}_n(\mathtt{x})$ n'a pas de solution. Mais, tous les s_i ainsi obtenus ne sont pas forcément des skypatterns pour M_u. Une étape de post-traitement doit être effectuée afin de les déterminer [6].

4. Expérimentations

Nous avons mené une évaluation expérimentale sur différents jeux de données de l'*UCI* et sur un jeu de données réel Mutagénicité [2], problème majeur dans l'évaluation des risques des substances chimiques fourni par le CERMN (`www.cermn.unicaen.fr`). Les résultats obtenus montrent l'efficacité de notre proposition et la qualité de nos règles de dérivation.

Conclusion. Nous avons conçu une méthode bottom-up efficace pour calculer le cube de skypatterns. Les expérimentations menées montrent l'intérêt de notre proposition. La navigation à travers le cube est une perspective très prometteuse.

Références

[1] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *ICDE*, pages 421–430, 2001.

[2] K. Hansen and al. Benchmark data set for in silico prediction of ames mutagenicity. *Journal of Chemical Information and Modeling*, 49(9) :2077–2081, 2009.

[3] C. Raïssi, J. Pei, and T. Kister. Computing Closed Skycubes. *PVLDB*, 3(1) :838–847, 2010.

[4] A. Soulet, C. Raïssi, M. Plantevit, and B. Crémilleux. Mining Dominant Patterns in the Sky. In *ICDM*, 2011.

[5] W. Ugarte, P. Boizumault, S. Loudni, and B. Crémilleux. Computing Skypattern Cubes. In *ECAI*, 2014.

[6] W. Ugarte, P. Boizumault, S. Loudni, B. Crémilleux, and A. Lepailleur. Mining (Soft-) Skypatterns using Dynamic CSP. In *CPAIOR*, pages 71–87, 2014.

Comprendre le Potentiel des Propagateurs

Sascha Van Cauwelaert[1] **Michele Lombardi**[2] **Pierre Schaus**[1]

[1] INGI, ICTEAM, Université Catholique de Louvain, Belgique
[2] DEIS, Université de Bologne, Italie
{sascha.vancauwelaert,pierre.schaus}@uclouvain.be michele.lombardi2@unibo.it

Abstract

Solvers performances can be improved through the cost reduction of propagators. The impact of this reduction has however an uncertain outcome. We propose a simple approach to estimate it in a realistic manner, before starting the actual research endeavor.

Résumé

Un axe de recherche important de notre communauté est la réduction du coût des propagateurs (algorithmes de filtrage de domaines), dans le but d'améliorer les performances des solveurs de contraintes. L'impact de cette réduction reste cependant incertaine. Nous proposons une approche simple permettant d'estimer celui-ci de façon réaliste, avant même de débuter le travail de recherche. Cet article résume le travail présenté dans [6].

Introduction La propagation est une caractéristique spécifique à la programmation par contraintes : elle permet à un solveur de contraintes de réduire l'espace de recherche, apportant parfois un important gain de performance. En pratique, la clé du succès réside dans un bon équilibre entre le temps sauvé en filtrant des valeurs et le temps passé à exécuter les propagateurs. Une quantité non-négligeable de travail de recherche vise à améliorer ce compromis.

Nous considérons le cas particulier pour lequel le but est d'optimiser la performance d'une technique de propagation donnée, sans changer son comportement du point de vue entrée-sortie. 2 façons d'obtenir cet effet sont 1) élaborer des algorithmes plus efficace fournissant le même filtrage ; 2) restreindre l'activation du propagateur en question à l'aide d'une condition nécessaire afin de réduire le nombre d'activations dénuées de filtrage. Pour 1) et 2), obtenir des améliorations est un long processus avec des résultats incertains. Dès lors, posséder des outils permettant de sonder le potentiel de techniques de propagation et d'évaluer l'impact d'améliorations spécifiques est d'un grand intérêt.

Le temps et le nombre d'échecs sont régulièrement utilisés pour comparer l'utilisation d'un propagateur avec une approche de base. Les 2 approches sont testées sur un ensemble d'instances résolues de manière complète (avec le risque de biaiser l'analyse vers des instances de tailles plus modestes). Aussi, des heuristiques de recherche statiques sont généralement choisies afin de réaliser des comparaisons équitables et rigoureuses. Ceci risque de réduire la portée de l'analyse car des heuristiques dynamiques sont généralement préférées en pratique. Nous proposons d'étendre ce type d'évaluation en : 1) instrumentant le solveur afin de collecter des informations au sujet de la contrainte ; 2) enregistrant et *rejouant* des arbres de recherches afin de permettre des comparaisons équitables avec des heuristiques et des tailles d'instances arbitraires. 2 cas d'étude sont ici considérés : le Raisonnement Énergétique [3, 1] et le Raisonnement en Cardinalité pour l'Empaquetage Revisité [5].

Approche Proposée Formellement, nous désirons évaluer une fonction filtrante ϕ qui lie un ensemble de domaines $D_0, \dots D_{n-1}$ à un second ensemble de domaines $D'_0, \dots D'_{n-1}$ tels que $D'_i \subseteq D_i$. Plus spécifiquement, nous voulons estimer le potentiel de deux pistes d'amélioration : 1) améliorer l'efficacité de l'implantation actuelle et 2) restreindre l'activation de ϕ à l'aide d'une condition nécessaire.

Mesure de Performance : Afin d'évaluer les performances de ϕ, nous comparons le temps pour résoudre un problème de satisfaction de contraintes avec et sans ϕ, dénotés respectivement par $M \cup \phi$ et M. Afin de s'assurer que 1) les 2 exécutions explorent le même espace de recherche et que 2) tous les nœuds de recherche sont visités dans le même ordre, nous proposons de « rejouer » l'arbre de recherche.

Technique de Rejeu : Le processus de recherche peut être vu comme l'évaluation de la fonction récursive $traverse(b, M)$ prenant en paramètre la stratégie de recherche b et le problème cible M. Soit $enregistre(b)$ et $rejoue(b)$ 2 stratégies de recherche d'emballage mémorisant et ré-ajoutant les contraintes retournées par b, respectivement. Pour évaluer ϕ, nous exécutons séquentiellement 1) $traverse(enregistre(b), M)$, 2)

$traverse(rejoue(b), M)$ et 3) $traverse(rejoue(b), M \cup \phi)$, et nous comparons les résultats de 2) et 3). Ceci permet 1) d'aborder des instances de tailles arbitraires et 2) d'utiliser n'importe quelle stratégie de recherche.

Évaluer le Potentiel du Propagateur : Soit t^+_ϕ et t^-_ϕ les temps d'exécution totaux de ϕ pour lesquels les activations ont respectivement conduit à de l'élagage de domaine ou non. Cette information est mesurée en introduisant une fonction d'emballage $stats(\phi)$ qui vérifie la taille des domaines avant et après l'exécution de ϕ. Si $t(b, M)$ est le temps requis pour exécuter $traverse(b, M)$, nous pouvons estimer l'impact de la réduction du temps d'exécution de ϕ par un facteur $\mu \in [0, 1]$ en calculant :

$$t(rejoue(b), M \cup stats(\phi)) - \mu \cdot (t^+_\phi + t^-_\phi)$$

De même, l'impact de la restriction de l'exécution de ϕ à l'aide d'une condition nécessaire qui empêche une fraction $\mu \in [0, 1]$ des activations inutiles peut être estimé en calculant :

$$t(rejoue(b), M \cup stats(\phi)) - \mu \cdot (t^-_\phi)$$

Cette simple approche linéaire nous permet de comparer des implantations réelles de ϕ avec des homologues fictifs, fournissant ainsi une chance d'explorer quelles valeurs de μ seraient nécessaires pour vaincre l'approche de base M, et donc une meilleure compréhension de l'effort requis pour arriver à une telle fin. Une plus grande sagacité peut être obtenue en opérant cette comparaison sur une panoplie de tests reconnue. Pour ce faire, nous nous reposons sur des *profils de performance* [2]. Ceux-ci sont des fonctions de distribution cumulatives $F(\tau)$ d'une métrique de performance donnée τ (dans notre cas le rapport entre le temps de résolution requis par une approche cible et celui de l'approche de base). En supposant la panoplie de tests assez représentative, $F(\tau)$ peut être interprétée comme une probabilité. Soit $\phi_0, \phi_1, \ldots$ l'ensemble des implantations de ϕ considérées, et $\mathcal{M}$ l'ensemble des instances de test. Le profil de performance de ϕ_i est donné par :

$$F_{\phi_i}(\tau) = \frac{1}{|\mathcal{M}|} \left| \left\{ M \in \mathcal{M} : \frac{t(rejoue(b), M \cup \phi_i)}{t(rejoue(b), M)} \leq \tau \right\} \right|$$

Pour une description sur la façon d'interpréter ces profils de performance, nous renvoyons le lecteur à l'article [6] que le présent document résume.

Expérimentations Nous avons appliqué notre approche à deux propagateurs, à savoir le *Raisonnement Énergétique* [3, 1] et le *Raisonnement en Cardinalité pour l'Empaquetage Revisité* [5]. Nous avons également considéré les classes de propagateurs fictifs suivantes : ϕ^{cost}_μ, i.e. une implantation pour laquelle le temps est réduit par un facteur μ ; $\phi^{cost}_{\mathcal{O}(f(n))}$, i.e. une implantation pour laquelle la complexité temporelle est $\mathcal{O}(f(n))$; ϕ^{oracle}_p, i.e. une implantation qui restreint l'exécution de ϕ à l'aide d'une condition nécessaire provoquant des activations inutiles avec une probabilité p.

Nous avons utilisé le solveur *OscaR* [4]. La configuration expérimentale, les résultats et leur analyse sont donnés dans [6]. Un exemple de profil de performance est donné dans la figure 1.

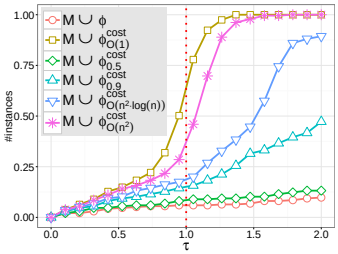


FIGURE 1 – Exemple de profil de performance pour le Raisonnement Énergétique.

Conclusion Évaluer les avantages potentiels fournis par la réduction du coût d'une procédure de filtrage est d'un grand intérêt afin de rendre nos efforts de recherche aussi fructueux que possible. De plus, mesurer exactement le gain obtenu par un algorithme de filtrage permet de réduire le biais des évaluations empiriques. Nous avons ici proposé une première étape dans cette direction, à savoir une méthodologie systématique permettant de simuler les performances d'implantations fictives de propagateurs avec un coût réduit.

Références

[1] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling : applying constraint programming to scheduling problems*, volume 39. Springer, 2001.

[2] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2) :201–213, 2002.

[3] Jacques Erschler and Pierre Lopez. Energy-based approach for task scheduling under time and resources constraints. In *2nd international workshop on project management and scheduling*, pages 115–121, 1990.

[4] OscaR Team. OscaR : Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`.

[5] François Pelsser, Pierre Schaus, and Jean-Charles Régin. Revisiting the cardinality reasoning for bin-packing constraint. In *Principles and Practice of Constraint Programming*, pages 578–586. Springer, 2013.

[6] Sascha Van Cauwelaert, Michele Lombardi, and Pierre Schaus. Understanding the potential of propagators. In *Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming (CPAIOR 2015)*, June 2015.

La première preuve d'optimalité pour le cluster de Lennard-Jones à cinq atomes

Charlie Vanaret[1] **Jean-Baptiste Gotteland**[2] **Nicolas Durand**[2] **Jean-Marc Alliot**[1]

[1] IRIT – ENSEEIHT, 2 Rue Charles Camichel, 31000 Toulouse

[2] ENAC – MAIAA, 7 Avenue Édouard Belin, 31055 Toulouse Cedex 04

charlie.vanaret@enseeiht.fr

Résumé

Le potentiel de Lennard-Jones est un modèle relativement réaliste décrivant les interactions entre deux atomes au sein d'un gaz rare. Déterminer la configuration la plus stable d'un cluster à N atomes revient à trouver les positions relatives des atomes qui minimisent l'énergie potentielle globale ; ce potentiel joue un rôle important dans le cadre des aggrégats atomiques et les nanotechnologies.

Le problème de cluster est NP-difficile et ouvert pour $N > 4$, et n'a jamais été résolu par des méthodes globales fiables. Nous proposons de résoudre le problème de cluster à cinq atomes de manière optimale avec des méthodes d'intervalles qui garantissent un encadrement du minimum global, même en présence d'arrondis. Notre modèle spatial permet d'éliminer certaines symétries du problème et de calculer des minorants plus précis dans le branch and bound par intervalles.

Nous montrons que la meilleure solution connue du problème à cinq atomes est optimale, fournissons la configuration spatiale correspondante et comparons notre solveur fiable aux solveurs BARON et Couenne. Alors que notre solution est numériquement certifiée avec une précision de 10^{-9}, les solutions de BARON et Couenne sont entachées d'erreurs numériques.

Abstract

The Lennard-Jones potential is a relatively realistic model that describes pairwise interactions between spherical noble gas atoms. Determining the most stable configuration of a cluster with N atoms amounts to finding the relative positions of the atoms that minimize the global potential energy ; this potential plays an important role in cluster chemistry and nanotechnologies.

The cluster problem is NP-hard and is open for $N > 4$. It has never been solved with reliable global optimization methods. We solve the cluster problem with five atoms using interval-based methods ; they compute a rigorous enclosure of the global minimum, even in the presence of roundoff errors. Our spatial model eliminates the symmetries of the problem and reduces the overestimation inherent to interval computations.

We show that the best known solution of the cluster problem with five atoms in the literature is optimal, and provide the corresponding spatial configuration. While our solution is numerically certified with precision 10^{-9}, the solutions of state-of-the-art solvers BARON and Couenne are vitiated by roundoff.

1 Motivation

1.1 Potentiel de Lennard-Jones

Le potentiel de Lennard-Jones [6] est un modèle relativement réaliste décrivant les interactions (répulsion à courte distance et attraction à grande distance) entre deux atomes sphériques au sein d'un gaz rare. Il est donné en unités réduites par :

$$V(d_{ij}^*) = 4\left(\frac{1}{{d_{ij}^*}^{12}} - \frac{1}{{d_{ij}^*}^{6}}\right) \tag{1}$$

où d_{ij}^* est la distance (en ångströms) entre les centres des atomes i et j.

Déterminer la configuration la plus stable d'un cluster à N atomes revient à trouver les positions relatives des atomes qui minimisent l'énergie potentielle globale :

$$\sum_{i<j}^{N} V(d_{ij}^*) = 4\sum_{i<j}^{N}\left(\frac{1}{{d_{ij}^*}^{12}} - \frac{1}{{d_{ij}^*}^{6}}\right) \tag{2}$$

Ce potentiel simplifié modélise les interactions de Van der Waals (terme attractif à la puissance 6) et joue donc un rôle crucial dans le *problème de repliement*

de protéine ; celui-ci consiste à prédire la structure tridimensionnelle d'une protéine à partir de sa séquence d'acides aminés.

1.2 Un problème ouvert

En dépit de son apparente simplicité, le problème de cluster de Lennard-Jones est NP-difficile ; les tests numériques de [8] suggèrent que le nombre de minima locaux croît de manière exponentielle avec N.

Pour $N \leq 4$, les positions des atomes qui minimisent l'énergie globale correspondent aux sommets d'un tétraèdre régulier. En revanche, le problème de cluster de Lennard-Jones pour $N \geq 5$ demeure un *problème ouvert* qui n'a jamais été résolu par des méthodes globales fiables [17]. Nombre de configurations *supposées optimales*[1] ont été obtenues grâce à un vaste panel de techniques de résolution approchées [10, 5, 7, 18, 4]. La meilleure solution connue pour le cluster de Lennard-Jones à cinq atomes est une bipyramide à base triangulaire [14], d'évaluation -9.103852415708.

2 Optimisation globale fiable

2.1 Analyse d'intervalles

L'analyse d'intervalles [9] est un outil de choix pour l'optimisation de problèmes à variables continues. Une *extension aux intervalles* F d'une fonction réelle f construit de manière automatique un minorant et un majorant rigoureux de f sur un intervalle $X = [\underline{X}, \overline{X}]$:

$$f(X) = \{f(x) \mid x \in X\} \subset F(X) \tag{3}$$

L'arithmétique d'intervalles est implémentée sur machine en arrondissant vers l'extérieur. Les algorithmes de branch and bound par intervalles [13] exploitent les propriétés conservatives de l'arithmétique d'intervalles pour borner le minimum global f^* de manière rigoureuse : $\tilde{f} - f^* < \varepsilon$, où $\tilde{f}$ est le meilleur majorant connu de f^*, et ε est la précision requise.

Cependant, les techniques d'intervalles produisent généralement une surestimation de l'image de la fonction en raison du *problème de dépendance* : les occurrences multiples des variables apparaissant dans l'expression analytique de f sont décorrélées. Par exemple, $X - X = [\underline{X} - \overline{X}, \underline{X} - \overline{X}]$ non réduit à $\{0\}$. L'ensemble construit est $\{x_1 - x_2 \mid x_1 \in X, x_2 \in X\} \supset \{x - x \mid x \in X\} = \{0\}$.

2.2 Programmation par contraintes sur intervalles

Les techniques de contraction [3] ont récemment donné un second souffle aux méthodes d'optimisation globale. Les *contracteurs* réduisent les domaines des variables par rapport à une contrainte ou un système de contraintes sans perte de solutions. En optimisation sans contraintes, ils peuvent être invoqués pour contracter les boîtes par rapport aux contraintes $f \leq \tilde{f}$ et $\nabla f = 0$ (où ∇f est le gradient de f).

En particulier, l'algorithme d'évaluation-propagation (aussi connu sous le nom HC4Revise [2]) calcule une approximation de la cohérence 2B. Il consiste à propager une contrainte en effectuant un double parcours de son arbre syntaxique, dans le but de contracter chaque occurrence des variables.

3 La première preuve d'optimalité pour cinq atomes

Nous proposons de résoudre le problème de cluster à cinq atomes avec des méthodes d'intervalles : notre algorithme Charibde [16] est un solveur hybride coopératif (figure 1) combinant un algorithme de branch and contract par intervalles et un algorithme stochastique à population [15]. Les deux algorithmes, indépendants, sont exécutés en parallèle et échangent bornes, solutions et espace de recherche par passage de messages. Charibde converge lorsqu'est trouvée une solution $\tilde{x}$ telle que $\overline{F(\tilde{x})} - f^* < \varepsilon$.

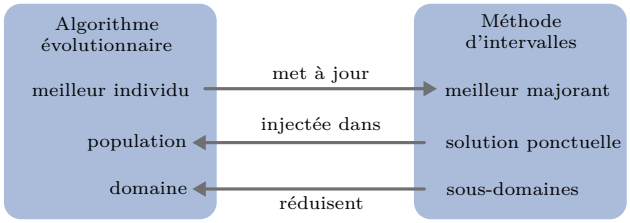


FIGURE 1 – Schéma coopératif de Charibde

Lorsque l'algorithme évolutionnaire améliore sa meilleure évaluation, il met à jour le meilleur majorant connu $\tilde{f}$ du minimum global afin d'accélérer les coupes du branch and contract par intervalles. En retour, lorsque la méthode d'intervalles trouve une solution ponctuelle qui améliore $\tilde{f}$, elle est injectée dans la population. Enfin, la population est périodiquement réinitialisée dans un domaine réduit, calculé à partir des sous-domaines restants. Ceci permet de ne pas explorer certains sous-espaces sous-optimaux éliminés par la méthode d'intervalles.

Charibde combine de manière séquentielle un algorithme de contraction de type HC4Revise et la différentiation automatique en mode adjoint, ce qui permet d'exploiter la contraction sur les nœuds intermédiaires dans l'arbre syntaxique [12].

1. Une liste des meilleures solutions connues est disponible sur `http://doye.chem.ox.ac.uk/jon/structures/LJ.html`

3.1 Modèle spatial

L'atome i est caractérisé par ses coordonnées cartésiennes $(x_i, y_i, z_i) \in \mathbb{R}^3$. On note $\boldsymbol{x} = (x_1, y_1, z_1, \ldots, x_N, y_N, z_N)$ le vecteur des $3N$ variables du cluster à N atomes. La distance interatomique $d_{ij}^* > 0$ entre les atomes i et j est alors :

$$d_{ij}^{*\,2} = (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \quad (4)$$

En injectant l'équation 4 dans l'équation 2, nous obtenons un problème d'optimisation sans contraintes.

3.2 Reformulation du problème

Une technique usuelle permettant d'éliminer le problème de dépendance dans l'équation 1 consiste à compléter le carré :

$$V(d_{ij}^*) = 4\left(\frac{1}{d_{ij}^{*\,6}} - \frac{1}{2}\right)^2 - 1 \quad (5)$$

L'expression analytique du potentiel de Lennard-Jones $V(d_{ij}^*)$ est donc optimale par rapport aux occurrences de d_{ij}^*. L'expression de la fonction objectif (équation 2) reste néanmoins sujette au problème de dépendance, car les coordonnées des atomes interviennent à plusieurs reprises dans les termes de distances d_{ij}^*. Notons que les coordonnées sphériques ou cylindriques feraient apparaître plusieurs occurrences des coordonnées dans chaque terme de distance, et n'ont donc pas été privilégiées.

L'équation 2 ne faisant intervenir que des *distances* entre atomes, la solution optimale du problème de cluster est invariante par translation et rotation. Afin d'éliminer les symétries et de réduire considérablement l'espace de recherche, les coordonnées de certains atomes peuvent être fixées :

$$\begin{cases} x_1 = y_1 = z_1 = 0 \\ x_2 \geq 0, y_2 = z_2 = 0 \\ x_3 \geq 0, y_3 \geq 0, z_3 = 0 \\ x_4 \geq 0, y_4 \geq 0, z_4 \geq 0 \end{cases} \quad (6)$$

La taille du problème de cluster à cinq atomes est alors réduite de 15 à 9 variables.

3.3 Solution optimale certifiée

Charibde prouve l'optimalité de la meilleure solution connue (voir sous-section 1.2) avec une précision $\varepsilon = 10^{-9}$:

$$f_5^* = -9.103852415707552 \quad (7)$$

Les coordonnées spatiales correspondantes sont données dans le tableau 1, pour un domaine initial $(x_i, y_i, z_i) \in [-1.2, 1.2]$. La configuration optimale est illustrée dans la figure 2.

Atome	x	y	z
1	0	0	0
2	1.1240936	0	0
3	0.5620468	0.9734936	0
4	0.5620468	0.3244979	0.9129386
5	0.5620468	0.3244979	-0.9129385

TABLE 1 – Solution optimale du cluster à cinq atomes

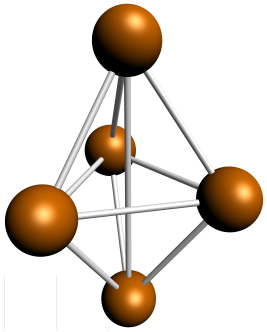


FIGURE 2 – Configuration optimale en bipyramide à base triangulaire

Les temps de calcul moyens et nombres d'évaluations (NE) moyens de la fonction objectif et de son gradient sont donnés dans le tableau 2 sur 100 exécutions. Charibde atteint l'optimum global f_5^* pour chacune des exécutions, après une moyenne de 764 itérations (en un temps de 0.11s), mais un total de 1436s est nécessaire à la preuve d'optimalité.

Temps CPU moyen (s)	1436
Temps CPU maximal (s)	1800
NE de f_5	483642320
NE de sa fonction d'inclusion	132 + 7088758
NE de son gradient	78229737

TABLE 2 – Résultats moyens sur 100 exécutions

3.4 Comparaison avec des solveurs de pointe

Les solveurs complets BARON [11] et Couenne [1] sont basés sur des algorithmes de branch and bound spatial ; les minorants de la fonction objectif sur les sous-domaines sont calculés par des relaxations convexes qui peuvent être sujettes à des erreurs d'arrondis. Une comparaison entre BARON, Couenne et Charibde (tableaux 3 et 4) suggère que les solveurs *non fiables* BARON et Couenne convergent beaucoup plus rapidement que les méthodes d'intervalles. Néanmoins, ils ne peuvent garantir de bornes sur la solution

en raison d'erreurs numériques, contrairement à Charibde. Ainsi, l'optimum *local* trouvé par BARON est incorrect à partir de la sixième décimale (pour une précision demandée de 10^{-9}) et l'optimum global de Couenne l'est à partir de la cinquième décimale (les décimales incorrectes sont soulignées).

Solveur	Minimum	Statut indiqué
BARON	-9.103853<u>46444055</u>	local
Couenne	-9.103870325603582	global
Charibde	-9.103852415707552	certifié

TABLE 3 – Minima de BARON, Couenne et Charibde

Solveur	Recherche	Preuve
BARON	0.23s	0.23s
Couenne	41.94s	61.7s
Charibde	0.11s	1436s

TABLE 4 – Comparaison des temps de convergence

4 Conclusion

Nous avons prouvé que la meilleure solution connue pour cinq atomes est optimale, fermant ainsi le problème ouvert de cluster de Lennard-Jones pour $N = 5$. Charibde fournit une preuve d'optimalité avec une précision $\varepsilon = 10^{-9}$ même en présence d'arrondis.

Notre modèle spatial élimine les symétries du problème et calcule des minorants plus précis la fonction objectif. Une comparaison de Charibde avec les solveurs de pointe BARON et Couenne montre que ces derniers ne peuvent garantir d'encadrement rigoureux du minimum global et produisent des solutions entachées d'erreurs numériques.

Références

[1] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4-5) :597–634, 2009.

[2] F. Benhamou, F. Goualard, L. Granvilliers, and J-F. Puget. Revising hull and box consistency. In *International Conference on Logic Programming*, pages 230–244. MIT press, 1999.

[3] G. Chabert and L. Jaulin. Contractor programming. *Artificial Intelligence*, 173 :1079–1100, 2009.

[4] Christodoulos A Floudas. *Deterministic global optimization*, volume 37. Springer, 1999.

[5] M.R. Hoare and P. Pal. Physical cluster mechanics : Statics and energy surfaces for monatomic systems. *Advances in Physics*, 20(84) :161–196, 1971.

[6] J. E. Jones. On the determination of molecular fields. i. from the variation of the viscosity of a gas with temperature. *Proceedings of the Royal Society of London. Series A*, 106(738) :441–462, 1924.

[7] R.H. Leary. Global optima of Lennard-Jones clusters. *Journal of Global Optimization*, 11(1) :35–53, 1997.

[8] Marco Locatelli and Fabio Schoen. Efficient algorithms for large scale global optimization : Lennard-Jones clusters. *Comput. Optim. Appl.*, 26(2) :173–190, 2003.

[9] R. E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[10] J. A. Northby. Structure and binding of Lennard-Jones clusters : 13 <= n <= 147. *The Journal of Chemical Physics*, 87(10) :6166–6177, 1987.

[11] Nikolaos V Sahinidis. Baron : A general purpose global optimization software package. *Journal of Global Optimization*, 8(2) :201–205, 1996.

[12] Hermann Schichl and Arnold Neumaier. Interval analysis on directed acyclic graphs for global optimization. *Journal of Global Optimization*, 33(4) :541–562, 2005.

[13] Stig Skelboe. Computation of rational interval functions. *BIT Numerical Mathematics*, 14(1) :87–95, 1974.

[14] N.J.A. Sloane, R.H. Hardin, T.D.S. Duff, and J.H. Conway. Minimal-energy clusters of hard spheres. *Discrete & Computational Geometry*, 14 :237–259, 1995.

[15] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, pages 341–359, 1997.

[16] C. Vanaret, J-B. Gotteland, N. Durand, and J-M. Alliot. Preventing premature convergence and proving the optimality in evolutionary algorithms. In *Proceedings of the Biennial International Conference on Artificial Evolution*, LNCS 8752. Springer, 2013.

[17] Stephen A. Vavasis. Open problems. *Journal of Global Optimization*, 4 :343–344, 1994.

[18] David J. Wales and Jonathan P. K. Doye. Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28) :5111–5116, 1997.

Backtracking distribué multi-variables avec sessions

Julien Vion **René Mandiau** **Sylvain Piechowiak**
LAMIH CNRS UMR 8201, UVHC, 59313 Valenciennes
{julien.vion, rene.mandiau, sylvain.piechowiak}@univ-valenciennes.fr

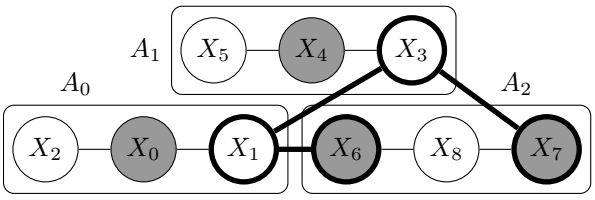
Résumé

Cet article est un résumé français de l'article *Multi-variable Distributed Backtracking with Sessions* [3]. Le formalisme des CSP Distribués (DisCSP) étend le modèle CSP classique pour représenter et résoudre des problèmes de décision ne pouvant être résolus de manière centralisée sur une seule machine, pour des raisons diverses (*e.g.*, taille excessive ou privacité). Dans cet article, nous proposons un algorithme complet de résolution de DisCSP, nommé *Backtracking distribué avec sessions* (DBS), qui a la particularité de gérer les DisCSP où chaque agent encapsule un sous-problème composé de plusieurs variables et contraintes. Nous prouvons que l'algorithme est correct et complet, et donnons des résultats expérimentaux prometteurs.

La majorité des travaux sur les DisCSP considèrent que chaque agent encapsule une seule variable. Cependant, la plupart des problèmes distribués (*e.g.*, emplois du temps, trafic routier, exploration multi-robots…) sont plus naturellement modélisés en affectant plusieurs variables et contraintes à chaque agent. Il est possible de décomposer le problème jusqu'à ce qu'il n'y ait plus qu'une seule variable par agent, ou encore de transformer le problème afin que le domaine de chaque variable représente l'ensemble des solutions au problème local, mais ces solutions ne sont guère satisfaisantes : elles ont entre autres pour effet d'augmenter exponentiellement le nombre de messages échangés ou la taille des domaines considérés.

Nous proposons l'algorithme *DBS*, qui se place dans la famille des algorithmes *ABT* [4, 1], avec la particularité de proposer une gestion spécifique des agents à plusieurs variables. D'autre part, pour établir le contexte des messages de backtrack, *ABT* nécessite de calculer un *nogood* lors de chaque conflit au cours de la recherche. Ces *nogoods* sont très coûteux à calculer [2]. Nous montrons qu'il est possible d'utiliser la notion de *session* pour définir le contexte des messages de backtrack, sans affecter les propriétés de correction et de complétude de l'algorithme. Ainsi, la prise en compte des messages est bien plus efficace. Cependant, il s'agit d'un compromis : en effet, les sessions sont moins porteuses d'information que les *nogoods* : il faut donc généralement échanger plus de messages pour résoudre un problème avec *DBS* qu'avec *(Multi-)ABT*. Nous réduisons l'impact de ces messages supplémentaires en proposant un ensemble de *filtres* permettant d'éliminer de nombreux messages inutiles sans affecter les propriétés de l'algorithme.

1 Fonctionnement de DBS



L'exemple ci-dessus est un problème de 2-coloration de graphe distribué entre trois agents A_0, A_1 et A_2, cités par ordre de priorité : *ABT* et *DBS* nécessitent de fixer un ordre de priorité statique entre les agents. Chaque agent choisit une solution locale (telle que représentée sur la figure), puis tente de l'étendre à ses voisins de priorité inférieure par l'envoi d'un message « OK ? ». Comme la solution de l'agent A_1 est incompatible avec la solution de A_0 (la contrainte $X_1 \neq X_3$ n'est pas respectée), A_1 va changer de solution pour avoir $X_3 =$ ●. L'agent A_2 reçoit trois messages : la solution locale de A_0, et les deux solutions locales successives de A_1. À chaque solution locale correspond un numéro de session. La première solution était consis-

tante et n'entrainait pas d'action de la part de A_2. La deuxième proposition va nécessiter pour A_2 de changer de solution, or il n'en existe pas qui soit compatible à la fois avec A_0 et A_1 : un message de backtrack est généré et est envoyé à A_1, accompagné du numéro de session correspondant.

Quand A_1 reçoit le message, il peut utiliser le numéro de session pour confirmer que la demande de backtrack correspond bien à sa solution courante (en effet, la première solution aurait pu être insatisfaisante pour A_2). Ici, *ABT* aurait nécessité des calculs complexes basés sur les nogoods attachés aux messages de backtrack. Comme A_1 n'a pas d'autre solution compatible, il transmet la demande de backtrack à A_0. Un processus similaire est alors executé pour la deuxième solution locale (symétrique) de A_0, qui détecte alors l'insatisfaisabilité du problème.

DBS utilise diverses structures de données pour mémoriser les solutions courantes des agents voisins de priorité supérieure ainsi que les solutions déjà proposées ou reçues : en effet, l'asynchronisme de l'algorithme peut amener un agent de priorité supérieure à proposer plusieurs fois une solution identique du point de vue de l'agent courant. Nous prouvons que *DBS* est correct et complet [3].

2 Gestion du multi-variable

Nous proposons une première optimisation simple pour gérer le multi-variable. Contrairement à *Multi-ABT* qui générait pour chaque agent une variable dont le domaine représentait l'ensemble des solutions locales, *DBS* ne considère que les solutions qui sont différentes du point de vue des autres agents (c'est-à-dire les affectations différentes des variables d'interface). Nous exploitons la notion de « voisinage partiellement interchangeable » (*PIN*).

D'autre part, *DBS* ne calcule pas l'ensemble des solutions au préalable, mais les génère au fur et à mesure de la recherche de manière « paresseuse ».

3 Filtrage des messages

Comme *DBS* tend à envoyer plus de messages que les autres algorithmes de la littérature, nous proposons un système de *filtres* (ou heuristiques) permettant d'éliminer certains messages obsolètes sans avoir à les traiter :

1. si un agent reçoit successivement deux propositions « OK ? » d'un même agent, seul le plus récent doit être considéré ;
2. à la réception d'un message « OK ? », tous les messages de backtrack en attente sont obsolètes ;
3. après l'envoi d'un message de backtrack, tous les messages de backtrack reçus des voisins de priorité inférieure à propos de la solution inconsistante deviennent obsolètes ;
4. si plusieurs messages « OK ? » sont en attente, il faut traiter en priorité les messages d'agents de priorité maximale pour réduire au mieux l'espace de recherche

4 Expérimentations & perspectives

Nous avons testé l'impact de l'utilisation des *PIN* et des différents filtres sur les performances de *DBS*, et l'avons comparé à d'autres algorithmes gérant le multi-variables : *Multi-ABT*, *Multi-AWC* et *AFC*.

Si le nombre de messages échangés par *DBS* peut rester important comparativement à ces algorithmes (jusqu'à deux ordres de magnitude avec une seule variable par agent), ceux-ci sont filtrés ou traités extrêmement rapidement, au point qu'aucune accumulation de messages n'est constatée dans la file d'attente. Dans le cas mono-variables, où *ABT* reste l'algorithme le plus performant, on constate que l'essentiel du temps de calcul de *DBS* est consacré à la gestion des messages, alors que pour ABT il est au niveau du calcul des nogoods. Quand le nombre de variables par agent augmente, *DBS* devient particulièrement performant (plus de deux ordres de magnitude en termes de *Wallclock CPU* de *NCCC*) par rapport aux autres algorithmes, dans la mesure où le nombre de messages n'explose plus.

Outre une amélioration possible des structures de données de *DBS*, nous prévoyons dans des travaux futurs d'étudier la gestion de la privacité et de la robustesse parmi cette famille d'algorithmes.

Références

[1] K. Hirayama, M. Yokoo et K. Sycara. "The Phase Transition in Distributed Constraint Satisfaction Problems : First results". In : *Proc. CP*. 2000, p. 515–519.

[2] D. L. Mammen et V. R. Lesser. "Problem Structure and Subproblem Sharing in Multi-Agent Systems". In : *Proc. ICMAS*. 1998, p. 174–181.

[3] R. Mandiau, J. Vion, S. Piechowiak et P. Monier. "Multi-variable Distributed Backtracking with Sessions". In : *Applied Intelligence* 41.3 (oct. 2014), p. 736–758.

[4] M. Yokoo. *Distributed Constraint Satisfaction : Foundations of Cooperation in Multi-agent Systems*. Springer Verlag, 2001.