Treizièmes Journées Francophones de Programmation par Contraintes

JFPC 2017

Montreuil-sur-mer, 13-15 juin 2017

JFPC 2017

Actes des Treizièmes Journées Francophones
de la Programmation par Contraintes

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

Organisation

CRIL, Université d'Artois

Présidents des Journées

Christophe Lecoutre, CRIL, Université d'Artois
Sébastien Tabary, CRIL, Université d'Artois

Président du comité de Programme

Frédéric Lardeux, LERIA, Université d'Angers

Comité de programme

Président

Frédéric LARDEUX (LERIA, Univ. d'Angers)

Membres :

- Gilles AUDEMARD (CRIL, Univ. Artois)
- Vincent BARICHARD (LERIA, Univ. d'Angers)
- Fabien CHHEL (Groupe ESEO)
- Thi-Bich-Hanh DAO (Univ. d'Orléans)
- Gilles DEQUEN (Univ. de Picardie Jules Verne)
- Pierre DESPORT (LERIA, Univ. d'Angers)
- Steven GAY (Google)
- Djamal HABET (LSIS, Univ. d'Aix-Marseille)
- Emmanuel HEBRARD (LAAS, Univ. de Toulouse)
- Marie-José HUGUET (LAAS, Univ. de Toulouse)
- Jean Marie LAGNIEZ (CRIL, Univ. Artois)
- Arnaud LALLOUET (Huawei Technologies)
- Nadjib LAZAAR (LIRMM, Univ. de Montpellier 2)
- Olivier LHOMME (IBM France)
- Xavier LORCA (Ecole des Mines de Nantes)
- Eric MONFROY (LINA, Univ. de Nantes)
- Samba Ndojh NDIAYE (LIRIS, Univ. Claude Bernard Lyon 1)
- Alexandre NIVEAU (GREYC, Univ. de Caen–Basse-Normandie)
- Alexandre PAPADOPOULOS (LIP6 UPMC, Sony CSL)
- Anastasia PAPARRIZOU (CRIL, Univ. Artois)
- Thierry PETIT (Worcester Polytechnique Institute, USA)
- Cédric PIETTE (CRIL, Univ. Artois)
- Jean-Charles REGIN (Univ. Nice-Sophia Antipolis)
- Mohamed SIALA (Insight Centre for Data Analytics, CS Dept., University College Cork)
- Laurent SIMON (Labri, Bordeaux Institute of Technology)
- Cyril TERRIOUX (LSIS, Univ. d'Aix-Marseille)
- Elise VAREILLES (Univ. Toulouse - Mines Albi)
- Sascha VAN CAUWELAERT (Univ. Catholique de Louvain)
- Nadarajen VEERAPEN (Univ. of Stirling)
- Philippe VISMARA (LIRMM, SupAgro)
- Mohamed WAHBI (Insight, University College Cork)

Rapporteurs Additionnels : Mehdi Maamar, Robin Arcangioli , Valentin Lemière, Anthony Palmieri, Maël Minot, Laure Devendeville, Lionel Paris.

Comité d'organisation

Présidents

— Christophe Lecoutre (CRIL, Univ. Artois)
— Sébastien Tabary (CRIL, Univ. Artois)

Membres :

- Gilles AUDEMARD (CRIL, Univ. Artois)
- Frédéric BOUSSEMART (CRIL, Univ. Artois)
- Guillaume CAVORY (Univ. Artois)
- Gaël GLORIAN (CRIL, Univ. Artois)
- Fred HÉMERY (CRIL, Univ. Artois)
- Yacine IZZA (CRIL, Univ. Artois)
- Sylvain MERCHEZ (Univ. Artois)
- Valentin MONTMIRAIL (CRIL, Univ. Artois)
- Anastasia PAPARRIZOU (CRIL, Univ. Artois)
- Eric PIETTE (CRIL, Univ. Artois)
- Cédric PIETTE (CRIL, Univ. Artois)
- Nicolas SZCZEPANSKI (CRIL, Univ. Artois)
- Yazid BOUMARAFI (CRIL, Univ. Artois)

Préface

Les Journées Francophones de Programmation par Contraintes (JFPC) sont maintenant depuis longtemps le rendez-vous annuel francophone où les chercheurs travaillant autour de la programmation par contraintes se retrouvent. Cette édition 2017 va de nouveau permettre d'échanger sur les dernières avancées du domaine.

Cette année le nombre d'articles soumis a été de 41. Nombre d'entre eux sont des traductions ou des résumés d'articles déjà acceptés dans des conférences très sélectives. Le taux d'acceptation élevé des JFPC (97% cette année) n'est donc pas représentatif de la qualité des articles car la sélection a été faite en amont dans les conférences d'origines des articles.

Il est a noter que cette année moins d'articles traitent du problème SAT alors que beaucoup plus d'articles s'intéressent à la modélisation. Nous observons aussi une augmentation du nombre d'articles cherchant à appliquer des résultats théoriques à des applications concrètes.

Lors de cette édition deux invités ont accepté de venir présenter certains de leur travaux. Élise Vareille abordera la manière dont elle utilise les contraintes au sein d'un configurateur interactif. De son côté, François Pachet nous parlera de la modélisation et de la génération de musique dans différents styles à l'aide d'algorithmes d'apprentissage.

Pour la seconde année consécutive un prix du meilleur article étudiant a été mis en place. L'objectif est de mettre en avant certains résultats de grande qualité obtenus par des doctorants lors de l'année écoulée.

Le comité de programme était composé de 32 membres. Ils ont été amené à faire des rapports sur tous les articles afin de permettre aux auteurs d'enrichir leur travail. Je sais que ce travail de relecture est long et peu valorisable mais pourtant ils ont tous fait un travail d'une très grande qualité. Je les en remercie très chaleureusement.

Pour terminer je tiens à remercier l'Association Française de Programmation par Contraintes (AFPC) qui par le biais d'un bureau très actif permet de dynamiser la communauté de Programmation par Contraintes. Enfin, cette édition 2017 n'aurait pas été possible sans l'énorme travail d'organisation de Christophe Lecoutre, Sébastien Tabary et de tout le comité d'organisation.

Frédéric Lardeux
Président du comité de programme des JFPC 2017

Table des matières

Ré-ordonnancement dynamique du trafic ferroviaire en cas de perturbations dans le réseau

Quentin Cappart **Pierre Schaus**

Université catholique de Louvain, Louvain-la-Neuve, Belgium
{quentin.cappart,pierre.schaus}@uclouvain.be

Résumé

Cet article résume le travail réalisé sur le ré-ordonnancement d'un trafic ferroviaire sujet à des perturbations pouvant survenir en temps réel dans le réseau (CPAIOR 2017 [3]). La contribution est une nouvelle façon de modéliser ce problème en utilisant la *programmation par contraintes* ainsi que les *variables conditionnelles d'intervalles de temps* (Conditional Time-Interval Variables). Les résultats montrent que l'utilisation de cette approche permet d'améliorer les décisions des opérateurs ferroviaires par rapport aux stratégies classiques de ré-ordonnancement, et cela, dans des conditions réalistes de fonctionnement.

1 Introduction

Depuis l'aube du 19^e siècle, le développement de l'industrie ferroviaire a pris une importance considérable dans la plupart des pays. D'années en années, le nombre de trains et la complexité des réseaux ne cesse d'augmenter. Dans ce contexte, le besoin d'un horaire efficace est crucial. En effet, un mauvais horaire peut engendrer des conflits entre les trains ainsi que des retards, ce qui se solde par des pertes financières et une diminution de la satisfaction des passagers.

En pratique, un horaire doit aussi être capable de faire face à des perturbations pouvant survenir en temps réel dans le réseau. Que ce soit des défaillances techniques, des accidents ou tout simplement les conséquences d'un horaire théorique trop optimiste, ce genre de perturbations peut engendrer des retards qui peuvent se propager dans le réseau. La conséquence directe est que l'horaire initial peut devenir infaisable et ainsi dégrader la situation. C'est pourquoi un ré-ordonnancement dynamique du trafic peut être nécessaire. Une série de travaux s'intéresse à cette problématique. De manière générale, la plupart des méthodes sont basées sur une *programmation mixte en nombre entiers*. Cependant, cette famille de méthodes pour résoudre les problèmes d'ordonnancement est connue pour être dépendante du choix de la granularité du temps.

Par ailleurs, l'enquête de Bartak et al. [1] montre que la *programmation par contraintes* est une bonne alternative pour résoudre les problèmes réalistes d'ordonnancement. Suivant cette tendance, Rodriguez [5] a développé un modèle pour résoudre le problème de ré-ordonnancement dynamique du trafic. Cependant, malgré les bonnes performances qu'il obtient, son modèle peut être amélioré par l'utilisation de contraintes globales. C'est ce que nous proposons de faire grâce à l'utilisation des *variables conditionnelles d'intervalles de temps* introduites par Laborie et al. [4]. Nos contributions sont les suivantes :

- Un modèle basé sur la *programmation par contraintes* et les *variables conditionnelles d'intervalles de temps* pour ré-ordonnancer le trafic ferroviaire face aux perturbations pouvant survenir dans le réseau. Les expériences effectuées montrent que l'approche proposée peut être utilisée pour résoudre les conflits dans des stations de grande envergure (Courtrai, en Belgique).
- La formulation d'une fonction objectif visant à la fois à minimiser le retard cumulé des trains et à maximiser la satisfaction des passagers tout en tenant compte de l'hétérogénéité du trafic.

2 Modélisation

L'objectif est de programmer adéquatement le déplacement des trains sur le réseau afin qu'ils atteignent leur destination. Cela implique à la fois de définir le chemin qu'ils devront prendre dans le réseau ainsi que

leur heure de départ. Ce problème peut être vu comme une variante d'un problème *d'ordonnancement d'ateliers à cheminements multiples* (Job-Shop Scheduling) où les ressources correspondent aux différentes portions du réseau ferroviaire et les activités aux déplacements des trains sur une portion du réseau. Les différences sont les suivantes :

- Certaines activités sont optionnelles. Elles peuvent ou non être exécutées dans l'horaire final. Cela vient du fait qu'un train peut choisir entre plusieurs chemins pour atteindre sa destination.
- La fin de certaines activités doit être synchronisée avec le début d'une autre. Cela modélise le fait que les trains ne disparaissent pas du réseau.
- Les activités peuvent utiliser plus d'une ressource. Cela modélise les réservations de plusieurs portions pouvant composer un chemin.

Variables de décision Notre modèle comporte des activités spécifiques, indiquant qu'un train suit un certain chemin. On en a une pour chaque correspondance entre un train et les chemins qu'il peut suivre. La décision se porte sur ces activités. Pour chacune d'entre elles on définit leur heure de départ et si elle est exécutée ou non. Une activité non exécutée indique que le train ne suivra pas ce chemin.

Contraintes Plusieurs contraintes sont présentes :

Précédence Lorsqu'un train suit un chemin, l'ensemble des activités composant le déplacement du train sur ce chemin doit être exécuté suivant un certain ordre.

Alternative Un train ne doit suivre qu'un et un seul un chemin.

Décomposition de chemins Une activité correspondant à un chemin peut être décomposée en sous-activités. Cette contrainte assure la cohérence de cette décomposition.

Ressource unaire Chaque portion du réseau ne peut être utilisée que par un seul train à la fois.

Consistance de l'ordre Les trains ne peuvent pas se dépasser dans le réseau.

Phase de recherche L'exploration de l'espace de recherche combine une *recherche orientée-échec* (Failure-Directed Search) avec une *exploration large du voisinage* (Large Neighborhood Search).

Fonction objectif Deux fonctions objectif sont considérées, une adaptée à un trafic homogène, et une autre, plus générale. Dans le premier cas, l'objectif est de minimiser la somme cumulée des retards des trains par rapport à l'horaire prévu. Dans le second, l'objectif est de minimiser la somme cumulée des retards des trains pondérée par le nombre de passagers dans chaque train. Par ailleurs, les trains appartiennent tous à une certaine catégorie de priorité. Il y a quatre catégories dans notre modèle. La fonction objectif est de type lexicographique où les optimisations successives se font suivant la catégorie des trains.

3 Résultats

Notre modèle a été comparé aux trois stratégies classiques de ré-ordonnancement : donner la priorité au premier train qui arrive, à celui ayant son heure d'arrivée prévue la plus proche, et finalement à celui ayant la plus haute priorité. Un temps de décision de trois minutes est fixé. Les résultats obtenus montrent que dans la quasi totalité des situations ($> 99\%$), notre approche améliore la meilleure solution obtenue avec les approches classiques. Dans le cas homogène, la réduction du retard moyen est de plus de 20% pour chacune des situations testées. Les résultats obtenus montrent que la *programmation par contraintes* avec les *variables conditionnelles d'intervalles de temps* peut être utilisée pour résoudre le problème de ré-ordonnancement dynamique du trafic.

Références

[1] Roman Barták, Miguel A Salido, and Francesca Rossi. New trends in constraint satisfaction, planning, and scheduling : a survey. *The Knowledge Engineering Review*, 25(03) :249–279, 2010.

[2] Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B : Methodological*, 63 :15–37, 2014.

[3] Quentin Cappart and Pierre Schaus. Rescheduling railway traffic on real time situations using time-interval variables. In *Fourteenth International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming*, 2017.

[4] Philippe Laborie and Jerome Rogerie. Reasoning with conditional time-intervals. In *FLAIRS conference*, pages 555–560, 2008.

[5] Joaquín Rodriguez. A constraint programming model for real-time train scheduling at junctions. *Transportation Research Part B : Methodological*, 41(2) :231–245, 2007.

Filtrage Efficace pour la contrainte Ressource-Coût Tous-Différents

Sascha Van Cauwelaert * **Pierre Schaus**

Pôle d'ingénierie informatique, Université Catholique de Louvain, Belgique
{sascha.vancauwelaert,pierre.schaus}@uclouvain.be

Résumé

Cet article résume le travail réalisé sur la contrainte Ressource-Coût Tous-Différents [1]. Elle permet de modéliser une série de problèmes pour lesquels un ensemble d'objets, chacun nécessitant une quantité d'énergie éventuellement différente, doivent être produits à des points de temps différents. L'objectif est de les planifier de telle sorte que la facture énergétique globale soit minimisée. La contrainte de calcul de coût peut être modélisée en Programmation Par Contraintes (PPC) soit en la décomposant en contraintes *élément*, soit en utilisant une contrainte globale *AffectationMinimum*. Malheureusement, la première approche obtient un faible filtrage et la seconde ne passe pas bien à l'échelle. Nous proposons donc un nouveau filtrage et l'évaluons sur un problème industriel réel. Nous montrons expérimentalement qu'il surpasse généralement les deux approches précédentes.

1 Introduction

Une conséquence de l'adoption des énergies renouvelables est une plus grande fluctuation des prix de l'énergie. Dans ce travail, nous considérons une famille de problèmes où les articles doivent être produits à des points de temps différents. Le coût énergétique est supposé être connu à chaque créneau temporel futur (fourni par un module de prévision du prix de l'électricité). L'objectif est alors de planifier chaque élément sur les créneaux horaires de sorte que la facture énergétique globale soit minimale. Pour chaque article, sa contribution au coût est l'énergie nécessaire pour la produire multipliée par la prévision de coût énergétique au moment où elle est produite.

Pour un ensemble donné d'éléments $\mathcal{I}$, nous pouvons formellement définir le coût total comme suit :

$$T = \sum_{i=1}^{|\mathcal{I}|} C(\mathcal{I}_i) * P(s(\mathcal{I}_i)) \quad (1)$$

où $s(\mathcal{I}_i)$ est le créneau attribué à l'élément $\mathcal{I}_i$, $C(\mathcal{I}_i)$ est sa consommation, et $P(t)$ est le coût de la ressource (i.e., l'énergie) à l'instant t.

En PPC, on peut modéliser ce coût de deux façons :

1. avec une somme de contraintes ÉLÉMENT.
2. avec une contrainte d'AffectationMinimum [**?**] qui a un filtrage basé sur des coûts réduits. Dans ce cas, le coût d'une arrête liant une variable (c'est-à-dire un objet) à une valeur (c'est-à-dire un créneau) est le produit de la consommation de l'objet avec le prix d'énergie au créneau.

Malheureusement, les deux approches ont des limites. La modélisation avec une somme de contraintes ÉLÉMENT ne prend pas en compte le fait que les éléments doivent tous être affectés à des créneaux différents. AffectationMinimum intègre la contrainte tous-différents, mais l'algorithme a une complexité assez élevée de $\mathcal{O}(n^3)$, où n est le nombre d'éléments.

Cet article propose une troisième approche en introduisant la contrainte Ressource-Coût Tous-Différents et un algorithme de filtrage qui passe à l'échelle. Son but est de calculer le coût total en traitant le fait que toutes les affectations doivent être différentes, tout en passant à l'échelle.

2 Borne Inférieure et Filtrage

Il y a un problème d'appariement inhérent au filtrage de Tous-Différents. Pour calculer efficacement notre borne inférieure, nous relâchons donc le

*Papier doctorant : Sascha Van Cauwelaert est auteur principal.

domaine des variables en supposant que toutes les variables peuvent être affectées à n'importe quelle valeur de n'importe quel domaine courant. Cependant, nous ne relâchons pas la contrainte tous-différents. Le problème d'appariement peut alors être résolu linéairement de manière gourmande sur le problème relaxé : il suffit de se baser sur les consommations et les coûts de ressource qui auront été triés préalablement avant la recherche. Le filtrage des domaines est également basé sur un pré-calcul exécuté à chaque appel du propagateur. L'implémentation repose sur plusieurs structures de données incrémentales et réversibles. Ceci nous permet d'obtenir une complexité temporelle pour vérifier toutes les paires de la variable-valeur en $\mathcal{O}(n.m)$, où n est le nombre de variables non-assignées et m est la taille du plus grand domaine de ces variables.

3 Evaluation

Pour évaluer notre approche en pratique, nous considérons un problème d'ordonnancement industriel réel, la Production Continue d'Acier avec Minimisation de la Facture d'Electricité (PCAMFE) (voir [1] pour sa définition exacte). Afin de présenter des résultats justes ne mesurant que les avantages dus à notre contrainte, nous avons suivi la méthodologie introduite dans [4, 3] et disponible dans le solveur *OscaR* [2]. Nous avons d'abord considéré des instances de petites tailles (96 créneaux). La Figure 1 fournit la comparaison du temps de résolution entre AffectationMinimum et Ressource-Coût Tous-Différents sur ces instances. Comme on peut le voir, la haute complexité temporelle ne permet pas à AffectationMinimum de résoudre ces instances efficacement. Pour des instances de grandes tailles (afin de tester le passage à l'échelle de notre algorithme), nous avons dès lors uniquement comparé notre approche avec la somme de contraintes élément. La Figure 2 illustre le profil de performance (produit avec l'outil [1] introduit dans [5]), que nous avons obtenu pour des instances de grandes tailles (480 créneaux). Comme on peut le voir, notre algorithme permet d'accélérer la résolution pour la plupart des instances (facteur 10 pour $\sim 20\%$ d'entre elles).

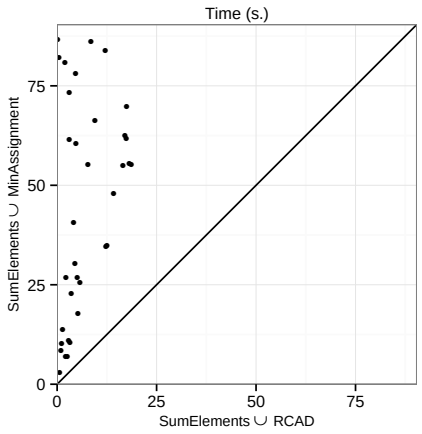


Figure 1 – Comparaison du temps de résolution entre AffectationMinimum et Ressource-Coût Tous-Différents sue des instances de petite taille.

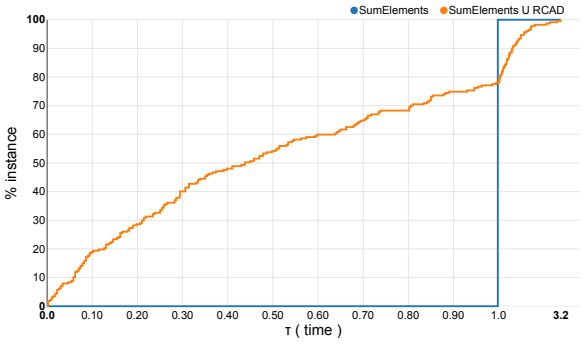


Figure 2 – Profile de performance pour des instances larges.

Références

[1] Sascha Van Cauwelaert and Pierre Schaus. Efficient filtering for the resource-cost alldifferent constraint. In *International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming*. Springer, 2017.

[2] 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.

[3] OscaR Team. OscaR : Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`.

[4] Sascha Van Cauwelaert, Michele Lombardi, and Pierre Schaus. Comprendre le potentiel des propagateurs. In *Actes des Onzièmes Journées Francophones de Programmation par Contraintes*, 2015.

[5] Sascha Van Cauwelaert, Michele Lombardi, and Pierre Schaus. Understanding the potential of propagators. In *Proceedings of the Twelfth International Conference on Integration of Artificial In-*

1. `http://performance-profile.info.ucl.ac.be/`

telligence and Operations Research techniques in Constraint Programming, 2015.

[6] Sascha Van Cauwelaert, Michele Lombardi, and Pierre Schaus. A visual web tool to perform what-if analysis of optimization approaches. Technical report, UCLouvain, 2016.

Filtrage efficace pour la Contrainte Disjonctive avec Temps de Transition regroupés par Famille

Sascha Van Cauwelaert * **Cyrille Dejemeppe** **Jean-Noël Monette** **Pierre Schaus**

Pôle d'ingénierie informatique, Université Catholique de Louvain, Belgique
{sascha.vancauwelaert,cyrille.dejemeppe,pierre.schaus}@uclouvain.be
jean-noel.monette@tacton.com

Résumé

Cet article résume le travail réalisé sur la contrainte disjonctive avec temps de transition regroupés par Famille [4]. Utilisée dans le cadre de problèmes d'ordonnancement, elle impose que des activités s'exécutant sur une même ressource ne se chevauchent pas, mais aussi qu'une durée minimale de transition soit présente entre elles. En particulier, les activités sont ici regroupées en familles : deux activités de la même famille ont une transition nulle l'une vers l'autre. Toutes les activités d'une famille donnée ont le même temps de transition vers toutes les activités d'une autre famille donnée. Cette particularité affaiblit le raisonnement réalisé par la contrainte disjonctive avec temps de transition [2, 5]. Ce travail propose une solution à ce problème en généralisant la contrainte. Les résultats montrent que notre approche permet le passage à l'échelle tout en conservant un élagage important.

1 Introduction

Les ressources disjonctives avec temps de transition pour des activités non-préemptives sont très fréquentes dans les problèmes d'ordonnancement de la vie réelle. Bien que des propagateurs efficaces aient été conçus pour la Contrainte Disjonctive (CD) standard [9], les contraintes de temps de transition entre les activités rendent généralement le problème plus difficile à résoudre parce que les propagateurs existants ne les prennent pas en compte. Nous avons récemment introduit dans [2, 5] un propagateur pour la Contrainte Disjonctive avec Temps de Transition (CDTT) comme extension des algorithmes de Vilím, afin de renforcer le filtrage en présence de temps de transition. Malheureusement, le filtrage supplémentaire diminue rapidement dans le cas d'une matrice de temps de transition creuse, ce qui se produit généralement lorsque les activités sont regroupées en *familles*, avec des temps de transition nuls au sein d'une famille.

La principale contribution de cet article est d'introduire des règles de filtrage et des structures de données adaptées pour tenir compte des familles. Le filtrage est testé expérimentalement sur des instances du problème d'Ateliers à Cheminements Multiples avec Temps de Transitions (ACMTT), bien qu'il puisse être utilisé pour tout type de problème. Les résultats montrent que notre propagateur améliore le temps de résolution par rapport aux approches existantes et passe bien à l'échelle.

2 Filtrage avec des Familles d'Activités

Dans la programmation par contraintes, un problème d'ordonnancement est modélisé en associant trois variables à chaque activité A_i : s_i, c_i, et p_i représentant respectivement le temps de début, de fin et de traitement de A_i. Ces variables sont reliées entre elles par la relation $\mathrm{s}_i + \mathrm{p}_i = \mathrm{c}_i$. Dans le contexte qui nous intéresse, un temps de transition $tt_{i,j}$ est une durée minimale qui doit se produire entre deux activités A_i et A_j si A_i précède A_j. Nous supposons que les temps de transition respectent l'inégalité triangulaire. La contrainte disjonctive avec contrainte de temps de transition impose la relation suivante (T est ici l'ensemble des activités s'exécutant sur la ressource) :

$$\forall A_i, A_j \in T : \\ A_i \neq A_j \implies (\mathrm{c}_i + tt_{i,j} \leq \mathrm{s}_j) \vee (\mathrm{c}_j + tt_{j,i} \leq \mathrm{s}_i)$$

*Papier doctorant : Sascha Van Cauwelaert est auteur principal.

Formellement, on note $F(A_i)$ la famille de A_i et par $\mathcal{F}$ l'ensemble de toutes les familles. Dans ce contexte, les temps de transition sont décrits comme une matrice carrée $\mathcal{TT}^{\mathcal{F}}$ de taille $|\mathcal{F}|$. Le temps de transition entre deux activités A_i et A_j est le temps de transition entre leurs familles respectives $F(A_i)$ et $F(A_j)$, et zéro si $F(A_i) = F(A_j)$:

$$\forall A_i, A_j \in T : tt_{i,j} = tt^{\mathcal{F}}_{F(A_i),F(A_j)}$$
$$\wedge \left(F(A_i) = F(A_j) \implies tt^{\mathcal{F}}_{F(A_i),F(A_j)} = 0\right)$$

Les algorithmes de propagation reposent tous sur un calcul efficace du Plus Tôt Temps d'Achèvement (PTTA) d'un ensemble d'activités Ω (écrit $ptta_\Omega$) à l'aide de structures de données appelées arbre-Θ et arbre-Θ-Λ [9]. Un des objectifs de notre travail était de conserver une faible complexité pour nos algorithmes, tout en prenant en compte la présence de familles afin de permettre un filtrage plus important que celui de [2, 5]. Le nouveau propagateur repose également sur un problème de plus court chemin mais sur un graphe sous-jacent différent. Le principal atout de notre approche est son passage à l'échelle : on obtient une quantité importante de filtrage tout en conservant une complexité en temps faible de $\mathcal{O}(n.\log(n).\log(f))$, pour n activités et f familles.

3 Évaluation

Afin de présenter des résultats justes ne mesurant que les avantages dus à notre contrainte, nous avons suivi la méthodologie introduite dans [7, 6] et disponible dans le solveur *OscaR* [3]. Nous nous sommes notamment comparés aux propagateurs d'Artigues et al. [1]. La Figure 1 fournit nos résultats sur les instances **t2ps** de l'ACMTT, avec un profil de performance (produit avec l'outil [1] introduit dans [8]). Comme on peut le voir, notre approche (courbe rouge) permet généralement de réduire le temps de résolution de façon importante.

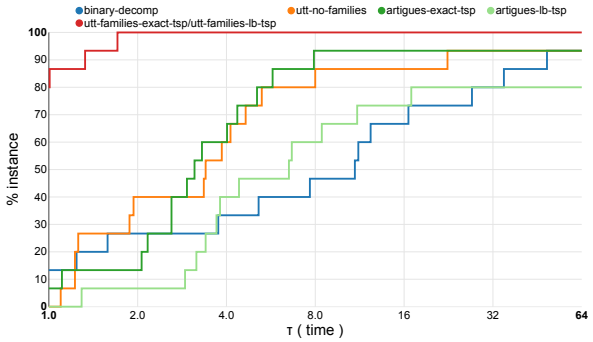


FIGURE 1 – Profils de performances sur les instances **t2ps** pour la métrique de temps.

Références

[1] Christian Artigues and Dominique Feillet. A branch and bound method for the job-shop problem with sequence-dependent setup times. *Annals of Operations Research*, 159(1) :135–159, 2008.

[2] Cyrille Dejemeppe, Sascha Cauwelaert, and Pierre Schaus. *Principles and Practice of Constraint Programming : 21st International Conference, CP 2015, Cork, Ireland, August 31 – September 4, 2015, Proceedings*, chapter The Unary Resource with Transition Times, pages 89–104. Springer International Publishing, Cham, 2015.

[3] OscaR Team. OscaR : Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`.

[4] Sascha Van Cauwelaert, Cyrille Dejemeppe, Jean-Noël Monette, and Pierre Schaus. Efficient filtering for the unary resource with family-based transition times. In *International Conference on Principles and Practice of Constraint Programming*, pages 520–535. Springer, 2016.

[5] Sascha Van Cauwelaert, Cyrille Dejemeppe, and Pierre Schaus. La contrainte disjonctive avec temps de transition. *JFPC 2016*, page 215.

[6] Sascha Van Cauwelaert, Michele Lombardi, and Pierre Schaus. Comprendre le potentiel des propagateurs. In *Actes des Onzièmes Journées Francophones de Programmation par Contraintes*, 2015.

[7] Sascha Van Cauwelaert, Michele Lombardi, and Pierre Schaus. Understanding the potential of propagators. In *Proceedings of the Twelfth International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming*, 2015.

[8] Sascha Van Cauwelaert, Michele Lombardi, and Pierre Schaus. A visual web tool to perform what-if analysis of optimization approaches. Technical report, UCLouvain, 2016.

[9] Petr Vilím. *Global Constraints in Scheduling*. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic, KTIML MFF, Universita Karlova, Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic, August 2007.

1. `http://performance-profile.info.ucl.ac.be/`

Des approches CP à la conquête de la théorie des bit-vecteurs *

Zakaria Chihani, Bruno Marre, François Bobot, Sébastien Bardin

CEA, LIST, Laboratoire de Sûreté Logiciel,Gif-sur-Yvette, France
(prenom.nom@cea.fr)

Résumé

Nous résumons ici l'article "Sharpening Constraint Programming approaches for Bit-Vector Theory", accepté à CPAIOR 2017, dans lequel nous exposons une nouvelle approche, CP(BV), basée sur la programmation par contraintes (CP) et appliquée à des problèmes dans la théorie des vecteurs de bits (BV). En prenant comme point de départ des avancées récentes et notre propre travail antérieur, nous explorons plus en profondeur et exploitons plus finement le cadre de communication naturel offert par CP. Nous montrons à travers des expérimentations poussées que notre approche peut rivaliser avec celle qui est plus couramment utilisée pour ce genre de problèmes : la satisfiabilité modulo théories (SMT).

1 Introduction

Il n'y a pas si longtemps, la vérification de programme était un but si ambitieux que même des esprits brillants la décrêtèrent "vouée à l'échec" [4] et arguèrent longuement sa stérilité. Ils conclurent en regrettant que, si malgré leurs arguments, "la vérification semble encore mériter qu'on l'explore, qu'il en soit ainsi". Et il en fut ainsi. Aujourd'hui la vérification est un domaine de recherche bien établi et son application en industrie n'est plus à démontrer.

Depuis le début des années 2000, la technique la plus usuelle consiste en la réduction de problèmes de vérification en des problèmes de SMT. Cette technique transforme le problème initial (BV) en un problème Booléen dépourvu de la structure originelle, qui aurait pu donner lieu à certains raccourcis et simplifications. Ainsi, bien que cette technique a, pour ainsi dire, fait ses preuves, elle demeure alourdie de quelques faiblesses que la CP peut pallier grâce à un traitement haut niveau, une propagation puissante et une communication fluide entre domaines.

Le papier accepté à CPAIOR 2017 [2], complémenté par un rapport technique disponnible en ligne, articule nos contributions autour des axes suivants : (1) la mise en place de la technique CP(BV), s'appuyant à la fois sur des résultats antérieurs et sur des idées nouvelles, notamment en matière de propagation inter-domaines, de simplification et factorisation de contraintes et de labelling ; (2) l'implémentation de cette technique dans une extension du solveur COLIBRI (en ECLiPSe Prolog) ; (3) l'évaluation systématique des différents aspects de notre technique ainsi que la comparaison avec les 5 meilleurs solveurs du domaine, selon le classement de la SMT-COMP : une compétition annuelle où des solveurs s'attaquent à des tests classifiés par théories, dont QF_BV pour les BV non-quantifiés, et QF_BVFP pour celle qui y mêle les flottants – et que notre solveur aurait remporté s'il y avait participé.

*Travail accompli en partie sous les projets ANR-14-CE28-0020 et ANR-12-INSE-0002. Le solveur CP COLIBRI est généreusement sponsporisé par l'IRSN

2 Motivation et intuition

Considérons les formules suivantes :

$$(A) : x \times y = (x\&y) \times (x|y) + (x\&\overline{y}) \times (\overline{x}\&y)$$
$$(B) : x_0 < x_1 < \cdots < x_n < x_0$$
$$(C) : \bigwedge_{i=0}^{n} (x_i < x_{(i+1)\%(n+1)} \ \& \ x_{(i+2)\%(n+1)})$$

où $\bar{\cdot}$ (resp. $\cdot \ \& \ \cdot$, $\cdot \mid \cdot$) est la négation (resp. conjonction, disjonction) bit-à-bit, $\wedge$ est le "et" logique, $<$ est l'operateur de comparaison non-signé et n a été fixé à 7.

La Table 1 montre le temps de résolution en secondes, selon la taille du vecteur choisie, de la preuve de la satisfiability de (A) et de l'insatisfiabilité de (B) et (C). TO indique que le solveur a été stoppé après 60-secondes

Table 1 – Comparaison (temps en sec.)

Formule	*taille(bits)*	*Z3*	*Yices*	*MathSAT*	*CVC4*	*Boolector*	*CP(BV)*
A	512	TO	1.60	6.04	17.28	20.55	0.24
	1024	TO	7.25	26.72	TO	TO	0.23
	2048	TO	31.83	TO	TO	TO	0.23
B	512	0.53	0.82	1.37	0	2.75	0.26
	1024	1.75	4.89	4.23	0	7.39	0.22
	2048	5.73	16.15	22.76	0	16.81	0.21
C	512	0.15	0.85	1.55	0.76	3.15	0.25
	1024	0.33	1.25	4.53	3.49	3.81	0.22
	2048	0.70	5.55	19.57	8.82	14.73	0.25

Le papier explique plus en détail la raison de la supériorité de l'approche CP(BV) sur les solveurs usuels en ce qui concerne ces exemples. Brièvement, cela tient de la collaboration et la communication (inter-réduction) de plusieurs représentations de domaines, assorties de leurs propagations, qui accomodent les différents types de contraintes. En l'occurrence, chaque variable est attachée à plusieurs représentations de domaines qui se complètent : des unions d'intervales et des congruences, efficaces sur l'arithmétique, des contraintes de différences globales, utiles pour les comparaisons, et enfin une abstraction du domaine bit-vecteur, inspirée par des traveaux précédents [1, 3], ainsi que des propagations associées.

3 Experimentations

Pour étayer nos contributions, nous évaluons notre méthode sur les dizaines de milliers de tests à la fois de nos partenaires industriels et de la SMT-COMP. Notre solveur résout environ 3/4 des benchmarks, ce qui est déjà supérieur aux autres approches CP similaires. De plus, nos expérimentations mettent en évidence plusieurs tests que notre solveur est le seul à passer, soulignant la complémentarité de l'approche CP(BV) avec les solveurs SMT. Nous nous sommes particulièrement inréressés à des sous-catéhories de tests non triviaux : ceux que les solveurs mettent plus de 5 secondes à résoudres, ceux qui contiennent de la combinaison de théories – notamment avec des flottants – et des problèmes avec des tailles de bit-vecteurs à 3, 4 ou 5 chiffres (pour souligner la mise à l'échelle – aspect qui pose problème en SMT). Nous évaluons également les différents choix de notre implémentation (simplifications, propagations, factorisations, communications) pour justifier l'utilité de ceux qui furent retenus.

4 Conclusion

Notre travail constitue une base solide sur laquelle des efforts furturs peuvent prendre appui pour remettre en question la croyance répandue selon laquelle la meilleure façon de traiter les BV est de les déléguer à un solveur SMT. En effet, il apparait clair que, malgré la non-optimalité (l'implémentation est codée en Prolog, il n'y a pas l'apprentissage dirigé par conflit qui fait la force des solveurs SMT), l'approche CP(BV) peut apporter un sérieux avantage en général et un boost particulier pour certaines classes de problèmes.

Références

[1] S. Bardin, P. Herrmann et F. Perroud. "An Alternative to SAT-Based Approaches for Bit-Vectors". In : *TACAS*. 2010.

[2] Z. Chihani et al. "Sharpening Constraint Programming approaches for Bit-Vector Theory". In : *to appear in CPAIOR*. 2017.

[3] L. D. Michel et P. Van Hentenryck. "Constraint Satisfaction over Bit-Vectors". In : *CP*. 2012.

[4] R. A. D. Millo, R. J. Lipton et A. J. Perlis. "Social Processes and Proofs of Theorems and Programs". In : *CACM* (1979).

Sharpening Constraint Programming approaches for Bit-Vector Theory*

Zakaria Chihani, Bruno Marre, François Bobot, Sébastien Bardin
CEA, LIST, Software Security Lab, Gif-sur-Yvette, France
(first.last@cea.fr)

Abstract

We address the challenge of developing efficient Constraint Programming-based approaches for solving formulas over the quantifier-free fragment of the theory of bitvectors (BV), which is of paramount importance in software verification. We propose CP(BV), a highly efficient BV resolution technique built on carefully chosen anterior results sharpened with key original features such as thorough domain combination or dedicated labeling. Extensive experimental evaluations demonstrate that CP(BV) is much more efficient than previous similar attempts from the CP community, that it is indeed able to solve the majority of the standard verification benchmarks for bitvectors, and that it already complements the standard SMT approaches on several crucial (and industry-relevant) aspects, notably in terms of scalability w.r.t. bit-width, theory combination or intricate mix of non-linear arithmetic and bitwise operators. This work paves the way toward building competitive CP-based verification-oriented solvers.

1 Introduction

Context. Not so long ago, program verification was such an ambitious goal that even brilliant minds decided it was "bound to fail" [37]. At the time, the authors concluded their controversial paper saying that if, despite all their reasons, " verification still seems an avenue worth exploring, so be it". And so it was. Today, software verification is a well established field of research, and industrial adoption has been achieved is some key areas, such as safety-critical systems.

Since the early 2000's, there is a significant trend in the research community toward reducing verification problems to satisfiability problems of first-order logical formulas over well-chosen theories (e.g. bitvectors, arrays or floating-point arithmetic), leveraging the advances of modern powerful SAT and SMT solvers [30, 43, 38, 6]. Besides weakest-precondition calculi dating back to the 1970's [19], most major recent verification approaches follow this idea [15, 25, 29, 35].

*Work partially funded by ANR under grants ANR-14-CE28-0020 and ANR-12-INSE-0002. The CP solver COLIBRI is generously sponsored by IRSN, the French nuclear safety agency.

The problem. While SMT and SAT are the *de facto* standard in verification, a few teams explore how Constraint Programming (CP) techniques can be used in that setting [33, 16, 26, 42, 27]. Indeed, CP could in principle improve over some of the well-known weaknesses of SMT approaches, such as non-native handling of finite domains theories (encoded in the Boolean part of the formula, losing the high-level structure) or very restricted theory combinations [39].

Yet, currently, there is no good CP-based resolution technique for (the quantifier-free fragment of) the theory of bitvectors [30], i.e. fixed-size arrays of bits equipped with standard low-level machine instructions, which is of paramount importance in verification since it allows to encode most of the basic datatypes found in any programming language.

Goal and challenge. We address the challenge of developing efficient Constraint Programming-based approaches for solving formulas over the quantifier-free fragment of the theory of bitvectors (BV). Our goal is to be able to solve many practical problems arising from verification (with the SMTCOMP challenge [1] as a benchmark) and to be at least complementary to current best SMT approaches. The very few anterior results were still quite far from these objectives [44], even if preliminary work by some of the present authors was promising on conjunctive-only formulas [1].

Proposal and contributions. We propose CP(BV), a highly efficient BV resolution technique built on carefully chosen anterior results [1, 36] sharpened with key original features such as thorough domain combination or dedicated labeling. Extensive experimental evaluations demonstrate that CP(BV) is much more efficient than previous similar attempts from the CP community, that it is indeed able to solve the majority of the standard verification benchmarks for bitvectors, and that it already complements the standard SMT approaches on several crucial (and industry-relevant) aspects, notably in terms of scalability w.r.t. bit-width, theory combination or intricate mix of non-linear arithmetic and bitwise operators. Our main contributions are the following:

- We present CP(BV), an original framework for CP-based resolution of BV problems, which built on anterior results and extend them with several key new features in terms of thorough domain combination or dedicated labeling. This results in a competitive CP-based solver, excelling in key aspects such as scalability w.r.t. bitwidth and combination of theories. A comparison of CP(BV) with previous work is presented in Table 1.
- We perform a systematic and extensive evaluation of the effect of our different CP improvements on the efficiency of our implementation. This compares the options at our disposal and justifies those we retained, establishing a firm ground onto which future improvements can be made. It also shows the advantage of our approach relative to the other CP approaches applied to BV, and that our approach is able to solve a very substantial part of problems from the SMTCOMP challenge.
- Finally, we perform an extensive comparison against the best SMT solvers for BV problems, namely: Z3 [8], Yices [20], MathSAT [14], CVC4 [4] and Boolector [11].

[1] `smtcomp.sourceforge.net`

This comparison exhibits the strenghts of CP over SMT approaches on particular instances. Specifically, our implementation surpasses several (and sometimes *all*) solvers on some examples involving large bit-vectors and/or combination with floating-point arithmetic.

Table 1: Overview of our method

	Bardin *et al* [1]	Michel *et al* [36]	CP(BV)
bitvector domain	+	++	++ [36]
arithmetic domain	++	+	++ [1]
domain combination	+	+	++
simplifications	+	x	++
BV-aware labeling	x	x	++
implemented	yes	no	yes
benchmark	conjunctive formulas		arbitrary formulas
	≈ 200 formulas		≈ 30,000 formula

Discussion. Considering that our current CP(BV) approach is far from optimal compared to existing SMT solvers (implemented in Prolog, no learning), we consider this work as an important landmark toward building competitive CP-based verification-oriented solvers. Moreover, our approach clearly challenges the well-accepted belief that bitvector solving is better done through bitblasting, opening the way for a new generation of word-level solvers.

2 Motivation

The standard (SMT) approach for solving bit-vector problem, called *bit-blasting* [7], relies on a boolean encoding of the initial bitvector problem, one boolean variable being associated to each bit of a bitvector. This *low-level encoding* allows for a direct reuse of the very mature and ever-evolving tools of the SAT community, especially DPLL-style SAT solvers [38, 43]. Yet, crucial high-level structural information may be lost during bitblasting, leading to potentially poor reasoning abilities on certain kinds of problems, typically those involving many arithmetic operations [12] and large-size bitvectors. Following anterior work [1, 36], we propose a *high-level encoding* of bitvector problems, seen as Constraint Satisfaction Problems (CSP) over finite (but potentially huge) domains. Each bitvector variable of the original BV problem is now considered as a (CSP) bounded-arithmetic variable, with dedicated domains and propagators.

Now illustrating with concrete examples, we show the kind of problems where our approach can surpass existing SMT solvers. Consider the three following formulas:

$$x \times y = (x \;\&\; y) \times (x \mid y) + (x \;\&\; \overline{y}) \times (\overline{x} \;\&\; y) \tag{A}$$

$$x_1 < x_2 < \cdots < x_n < x_1 \tag{B}$$

$$(\bigwedge_{i=1}^{n-2} x_i < x_{i+1} \;\&\; x_{i+2}) \wedge (x_{n-1} < x_n \;\&\; x_1) \wedge (x_n < x_1 \;\&\; x_2) \tag{C}$$

where $\bar{\cdot}$ (resp. $\cdot$ & $\cdot$, $\cdot \mid \cdot$) is the bit-wise negation (resp. conjunction, disjunction), $\wedge$ is the logical conjunction, $<$ is an unsigned comparison operator, n was chosen to be 7. As an example, the SMT-LIB language [5] encoding of formula A is:

```
(assert (= (bvmul X Y) (bvadd (bvmul (bvand X Y) (bvor X Y))
(bvmul (bvand X (bvnot Y)) (bvand (bvnot X) Y)))))
```

Table 2 shows the time in seconds according to bit-vector size, both for the satisfiability proof of the *valid* formula A and the unsatisfiablity of B and C. CP(BV) is the name of our technique, and TO means that the solver was halted after a 60-second timeout.

Table 2: Comparison of performance (time in sec.) for different solvers

Formula	size(bits)	Z3	Yices	MathSAT	CVC4	Boolector	CP(BV)
A	512	TO	1.60	6.04	17.28	20.55	0.24
	1024	TO	7.25	26.72	TO	TO	0.23
	2048	TO	31.83	TO	TO	TO	0.23
B	512	0.53	0.82	1.37	0	2.75	0.26
	1024	1.75	4.89	4.23	0	7.39	0.22
	2048	5.73	16.15	22.76	0	16.81	0.21
C	512	0.15	0.85	1.55	0.76	3.15	0.25
	1024	0.33	1.25	4.53	3.49	3.81	0.22
	2048	0.70	5.55	19.57	8.82	14.73	0.25

On these examples, CP(BV) clearly surpasses SMT solvers, reporting no TO and a very low (size-independent) solving time. In light of this, we wish to emphasize the following advantages of our CP(BV) high-level encoding for bitvector solving:

- Each variable is attached to *several and complementary domain representations*, in our case: *intervals plus congruence*, *bitlist* [1] (i.e. constraints on the possible values of specific bits of the bitvector) and *global difference constraint* (delta). Each domain representation comes with its own *constraint propagation* algorithms and deals with different aspects of the formula to solve. We will call *integer domains* or *arithmetic domains* those domains dealing mainly with high-level arithmetic properties (here: intervals, congruence, deltas), and *BV domains* or *bitvector domains* those domains dealing with low-level aspects (here: bitlist).

- These domains can *freely communicate between each other*, each one refining the other through a *reduced product* (or *channeling*) mechanism [41]. For example, in formula B, adding the difference constraint to the delta domain does allow CP(BV) to conclude *unsat* directly at propagation. Having multiple domains also allows to search for a solution in the smallest of them (in terms of the cardinality of the concretization of the domain abstraction).

- In the case of formula C, a reduced product is not enough to conclude at propagation. Here, a BV constraint *itself* refines an arithmetic domain: indeed, with the simple observation that, if a & $b = c$ then $c \leqslant a$ and $c \leqslant b$, the bit-vector part of CP(BV) not only acts on the bit-vector representation of the variables, but also "informs" the global difference constraints of a link it could not have found on its own.

3 Background

This section lays down the ground on which our research was carried out, both the theoretical foundations, anterior works and the COLIBRI CP solver [33].

3.1 BV theory

We recall that BV is the quantifier-free theory of bitvectors [30], i.e. a theory where variables are interpreted over fixed-size arrays (or vectors) of bits along with their basic operations: logical operators (conjunction " & ", disjunction "|" and xor "$\oplus$", *etc.*), modular arithmetic (addition $+$, multiplication $\times$, *etc.*) and other structural manipulations (concatenation :: , extraction $\lfloor . \rceil_{i.j}$, etc.).

3.2 CP for Finite-Domain Arithmetic

A *Constraint Satisfaction Problem* [18] (CSP) consists in a finite set of variables ranging over some domains, together with a set of constraints over these variables – each constraint defining its own set of solutions (valuations of the variables that satisfy the constraint). Solving a CSP consists in finding a solution meeting all the constraints of the CSP, or proving that no such solution exists. We are interested here only in the case where domains are finite. *Constraint Programming* [18] (CP) consists in solving a CSP through a combination of *propagation* and *search*. Propagation consists mainly in reducing the potential domains of the CSP variables by deducing that some values cannot be part of any solution. Once no more propagation is possible, search consists in assigning a value to a variable (taken from its reduced domain) and continue the exploration, with backtrack if required.

The CP discipline gets its strenght from global constraints and capabilities for dense interreductions between domain representations, along with constraint solving machinery. We present in this section standard domains and constraints for bounded arithmetic.

Union of intervals. A simple interval $[a; d]$, where $a, d \in N$ represents the fact that a given variable can take values only between a and d. A natural extension of this notion is the *union of intervals* (*Is*). As a shortened notation, if $x \in \{a\} \uplus [b; c] \uplus \{d\}$, one writes $[x] = [a, b{\cdot\cdot}c, d]$.

Congruence [31]. If the division remainder of a variable x by a divisor d is r (*i.e.,*x satisfies the equation $x\%d = r$), then $\langle x \rangle = r[d]$ is a *congruence* and represents all values that variable x can take, *e.g.,* $\langle x \rangle = 5[8]$ means $x \in \{5, 13, 21, \ldots\}$.

Global difference constraint (Delta) [23]. A global difference constraint is a set of linear constraints of the form $x - y \diamond k$ where $\diamond \in \{=, \neq, <, >, \leqslant, \geqslant\}$. Tracking such sets of constraints allows for better domain propagation and early infeasibility detection, thanks to a global view of the problem compared with the previous (local) domains.

3.3 The COLIBRI Solver for FD Arithmetic

The COLIBRI CP solver [33] was initially developped to assist CEA verification tools [9, 47, 2, 17]. COLIBRI supports bounded integers (both standard and modular arithmetic [28]), floating-points [34] and reals. Considering arithmetic, COLIBRI already provides all the domains described in Section 3.2, together with standard propagation techniques and strong interreductions between domains. Search relies mostly on a standard fail-first heuristics. COLIBRI is implemented in ECLiPSe Prolog, yielding a significant performance penalty (compared with compiled imperative languages such as C or C++) but allowing to quickly prototype new ideas.

3.4 Former CP(BV) approaches

Two papers must be credited with supplying the inspiration for this work, written by Bardin et al [1] and by Michel and Van Hentenryck [36]. Put together, these two papers had good ideas, which we adopted, unsatisfactory ideas which were disgarded, and finally ideas that were not advanced enough which we extended.

The first paper [1] introduces the bitlist domain, i.e. lists of four-valued items ranging over $\{0, 1, ?, \bot\}$ – indicating that the i^{th} bit of a bitvector must be 0, 1, any of these two values (?), or that a contradiction has been found ($\bot$) – together with its propagators for BV operators. Moreover, the authors also explain how arithmetic domains (union of intervals and congruence) can be used for BV, and describe first interreduction mechanisms between bitlists and arithmetic domains.

The second paper [36] introduces a very optimized implementation of bitlists, using two BVs $\langle {}^1x, {}^0x\rangle$ to represent the bitlist of x, where ${}^1x$ (resp. ${}^0x$) represents bits known to be set (resp. cleared) in x. The efficiency comes from the use of machine-level operations for performing domain operations, yielding constant time propagation algorithms for reasonable bitvector sizes.

Basically, we improve the technique described in [1] by: borrowing the optimized bitvector domain representation from [36] (with a few very slight improvements), significantly improving inter-domain reduction, and designing a BV-dedicated search labeling strategy.

As improving inter-domain reduction is one of our key contributions, we present hereafter the reductions between BV domains and arithmetic domains described in [1]:

With congruence: the BV domain interacts according to the longest sequence of known least significant bits. For example, a BV domain $[\![10?00?101]\!]$ of a variable b indicates that b satisfies the equation $b[8] = 5$, which therefore constrains the congruence domain using $5[8]$. Conversely a known congruence of some power of 2 fixes the least significant bits.

With *Is*: for a variable x, the union of intervals can refine the most significant bits of the BV domain by clearing bits according to the power of two that is immediately greater than the maximum extremum of the *Is*. And the BV domain influences *Is* by (only) refining the extrema through the maximum and minimum bit-vectors allowed, and by removing *singletons* that do not conform to the BV domain.

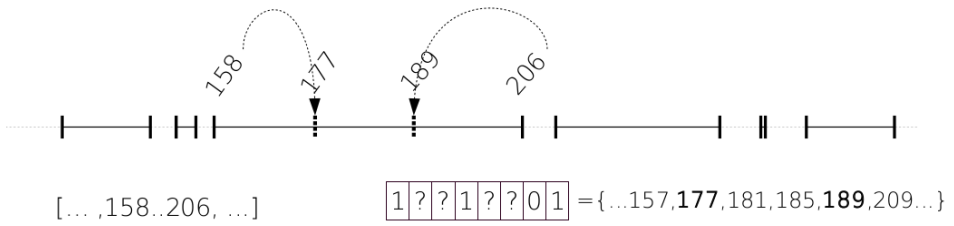


Figure 1: Enlarging the gaps in the *Is* according to the BV domain

4 Boosting CP(BV) for Efficient Handling of Bit-Vectors

In this section, we delve into the specificities of our approach, leaving complex details to a technical report[2]. We first start by presenting a significantly improved inter-domain reduction, followed by new reductions from BV constraints to other domains, then we show some simplifications and factorisations at the constraint level, and we finish by presenting our BV-dedicated labeling strategy.

4.1 Better interreduction between BV- and arithmetic- domains

We present here several significant improvements to the inter-domain reduction techniques proposed in [1] between bitlists and unions of intervals. Our implementation also borrows the inter-reduction between bitlists and congruence from [1].

***Is* to BVs.** Let m and M be respectively the minimal and the maximal value of a union of intervals. Then, the longest sequence of most-significant bits on which they "agree" can also be fixed in the bit-vector domain. For example, $m = 48$ and $M = 52$ (00110000 and 00110100 in binary) share their five most-significant bits, denoted $[\![00110???]\!]$. Therefore, a bit-vector $bl = [\![0??1???0]\!]$ can be refined into $[\![00110??0]\!]$. For comparison, the technique in [1] only reduces bl to $[\![00?1???0]\!]$.

BV to *Is*. Consider a variable b with a *Is* domain $[b] = [153, 155, 158..206, 209]$, and a bit-vector domain $(\!|b|\!) = [\![1??1??01]\!] = \{\cdots, 153, 157, 177, 181, 185, 189, 209, \cdots\}$, as illustrated in Figure 1. The inter-domain reduction from [1] can refine the extremum of the *Is* (here: nothing to do, since 153 and 209 both conforms to $(\!|b|\!)$) and removes the singletons that are not in $(\!|b|\!)$ (here: 155), yielding $[b] = [153, 158..206, 209]$. We propose to go a step further by refining each bound inside a *Is*, such that after reduction each bound of $[b]$ conforms to $(\!|b|\!)$. Here, 158 (resp. 206) is not allowed and should be replaced by its closest upper (resp. lower) value in $(\!|b|\!)$, i.e. 177 (resp. 189), yielding $[b] = [153, 177..189, 209]$.

We have designed such a correct and optimal reduction algorithm from bitlist to *Is*. Since we work on the $\langle {}^1x, {}^0x\rangle$ representation of bitlists, the algorithm relies on machine-level operations and is linear in the size of the bitvector (cf. technical report). We describe the procedure for increasing a lower bound in Alg. 1; de-

[2] `sites.google.com/site/zakchihani/cpaior`

creasing the upper bound (symmetrically) follows the same principle (cf. technical report). In order to calculate a new bound r accepted by the bit-vector domain $\langle\!|b|\!\rangle$, we start by imposing on the lower bound l what we already know, *i.e.,*set what is set in ${}^1b$ and clear what is cleared in ${}^0b$ (line 1 of Alg 1). Then flag the bits that were changed by this operation, going from cleared to set and from set to cleared. To refine the lower bound, we must raise it as much as necessary but not one bit higher, *i.e.,*we should look for the smallest amount to add to the lower bound in order to make it part of the concretisation of $\langle\!|b|\!\rangle$. This entails two things: a cleared bit can only become set if all bits of lower significance get cleared. For example, to increase the binary represented integer **0**10 exactly until the left-most bit gets set, we will pass by 011 and *stop* at **1**00 : going to 101 would increase more than strictly necessary. Similarly, the *smallest* increase that clears a set bit i is one where the *first cleared bit on the left* of i can be set (line 12 of Alg 1, function `left-cl-can-set-of`). For example, to clear the third most significant bit (in bold) in 01**1**011, one needs to increase to 01**1**100, 01**1**101, 01**1**110, 01**1**111 then reach 10**0**000. Doing so clears not only the target bit i but all the bits of lower significance.

Algorithm 1 Increasing the lower bound l according to $\langle\!|b|\!\rangle$

```
1: r := ¹b | l & ⁰b
2: set2cl:= l & r̄
3: cl2set:= l̄ & r
4: if cl2set > set2cl then
5:     size:=log₂(cl2set)
6:     mask0 := −1 ≪ size
7:     can-cl:= mask0 | ¹b
8:     r := r & can-cl
9: else
10:    size:=log₂(set2cl)
11:    cl-can-set:= r̄ & ⁰b
12:    next-to-set := left-cl-can-set-of(size, cl-can-set)
13:    r :=set(r,next-to-set)
14:    mask0 := −1 ≪next-to-set
15:    can-cl:= mask0 | ¹b
16:    r := r & can-cl
17: end if
```

Drilling the *Is* according to BV. If the *Is* contains only one interval, then our technique does not improve over [1], and is only slighly superior to the channeling method of [36]. For this reason, we force the bit-vector domain to create at least one gap in the union of intervals. Consider for example a domain $bl = [\![0?10?1?]\!]$. When observing the concretisation $\{18, 19, 22, 23, 50, 51, 54, 55\}$, the largest gap is between 23 and 50, *i.e.,*0010111 and 0110010, obtained by fixing the *most significant unknown bit (msub).* More generally, for a variable x the largest gap is created by intersecting $[x]$ with $[{}^1x_{\cdot\cdot}a, b_{\cdot\cdot}{}^0x]$, where a is obtained by clearing the *msub* and setting all other unknown bits, and b is obtained by setting the *msub* and clearing all other unknown bits. One can of course enforce more gaps, but there is a tradeoff between their propagation cost (as *Is*) and their benefits. In this work, using one gap was satisfactory.

4.2 BV constraints reducing arithmetic domains

Our CP(BV) approach strives to keep each of its domains as aware as possible of the other domains. We now show how non-BV domains can be reduced through BV constraints. In the following, we recall that $[x]$ denotes the union of intervals attached to variable x.

4.2.1 BV constraints on *Is*

It turns out that most BV constraints can influence unions of intervals.

Bitwise binary operands: a disjunction $x \mid y = z$ can refine $[z]$ in more than one way, but experimentation showed a notable effect only when $[x]$ and $[y]$ contain only singletons, at which case $[z]$ can be refined by the pairwise disjunction of those singletons. Similar refinements can occure through the bitwise conjunction and exclusive disjunction. For the latter, one can also refine in the same manner the *Is* of the operands, since $x \oplus y = z$ implies the same constraint for all permutation of x, y, z.

Negation: from a negation constraint $x = \overline{y}$, one can refine the *Is* of one variable from that of the other. By mapping each singleton $\{c\} \in [x]$ and interval $a..b \in [x]$ to $\{\overline{c}\}$ and $\overline{b}..\overline{a}$, we build a *Is* to populate $[y]$. The symmetric construction populates $[x]$.

Shifts: from a right shift $x \gg y = z$, which is equivalent to a natural division (*i.e.*,$x/2^y = z$), one can refine $[z]$ simply by right-shifting all elements of $[x]$ (singletons and bounds of internal intervals) by y. The left-shift constraint is treated mostly in the same way but requires extra care as it is a modular multiplication and it can overflow.

Sign-extension: when extending the sign of x by i positions to obtain z, the method consists in splitting the $[x]$ by the integers that are interpreted as negative, most significant bit is 1, and the one interpreted as positive, most significant bit is 0 and to apply the sign extention separately, disjunction with $2^i - 1 \ll i$ for the firsts and identity for the seconds.

Extractions: when extracting from the left-most to any position, it's the same as a right logical shift. The more general case is tricky. Take $\lfloor x \rceil_{i..j} = y$ to mean the extraction from bit i to j of x to obtain y (with $\|x\| > i \geqslant j \geqslant 0$), then a singleton in for an interval $x_a..x_b$,

- If $(x_b \gg j) - (x_a \gg j) > 2^{i-j}$, then the interval necessarily went through all integer coded on i bits, so the integer domain cannot be refined.
- else, if $(x_b \oplus x_a)$ & $2^i = 0$, then no power of 2 was traversed, the bounds can simply be truncated and stay in that order: $(\lfloor x_a \rceil_{i..j})..(\lfloor x_b \rceil_{i..j})$
- else, 2^i was traversed, then the *Is* is $[0..(\lfloor x_b \rceil_{i..j}), (\lfloor x_a \rceil_{i..j})..(2^{(i-j)} - 1)]$

For example, using the binary representation for the integer bounds of a union of intervals, an extraction of the 3 rightmost bits of a variable whose union of intervals contains 01110..10011 would not produce the invalid interval 110..011 because its lower bound is greater than its upper bound. This falls in the third case above, and would generate the two intervals 000..011 and 110..111.

Concatenation: for a concatenation $x :: y = z$, the inner structure of $[z]$ can be refined from $[x]$ and $[y]$. Let $v^{\ll x}$ be $v \mid (x \ll \|y\|)$ and $(a{\cdot\cdot}b)^{\ll x}$ be $a^{\ll x}{\cdot\cdot}b^{\ll x}$. For example, if $[x] = [x_a{\cdot\cdot}x_b]$ and $[y] = [y_1, y_2{\cdot\cdot}y_3, y_4{\cdot\cdot}y_5]$, then $[z]$ can be refined by $[(y_1)^{\ll x_a}, (y_2{\cdot\cdot}y_3)^{\ll x_a}, (y_4^{\ll x_a}){\cdot\cdot}(y_1^{\ll x_b}), (y_2{\cdot\cdot}y_3)^{\ll x_b}, (y_4{\cdot\cdot}y_5)^{\ll x_b}]$. The algorithm is described in the technical report. One can also refine $[x]$ and $[y]$ from $[z]$.

4.2.2 BV constraints on deltas

As seen in the motivation section, keeping the deltas informed of the relationship between different variables can be an important factor for efficient reasoning.

Bitwise operations: a constraint $x \mid y = c$ implies that that $(c - y \leqslant x \leqslant c) \wedge (c - x \leqslant y \leqslant c)$. Symmetric information can be derived for conjunction. Exclusive disjunction, however, does not derive knowledge regarding deltas. The bitwise negation has limited effect and can only impose that its argument and its result be different.

Extraction: regardless of the indices, the result of an extraction is always less than or equal to its result. As a matter of fact, an extraction $\lfloor x \rceil_{i.j} = (x \% 2^i)/2^j$ and can enjoy the same propagations on the non-BV domains.

More generally: many BV constraints can be mapped to an integer counterpart and propagate on non-BV domains. For example, a concatenation $x :: y$ can have the same effect as the (overflowless) integer constraint $z = x \times 2^{\|y\|} + y$ would.

4.3 Factorizations and simplifications

In the course of solving, a constraint can be simplified or become duplicate or a subsumption of another constraint. These shortcuts can be separated in two categories.

Simplifications. Neutral and absorbing elements provide many rewriting rules which replace constraints by simpler ones. In addition to these usual simplifications one can detect more sophisticated ones, such as if $z \gg y = z$ and $y > 0$ then $z = 0$ (without restricting the value of y), and when z is x rotated i times, if the size of x is 1 or if $i \% \|x\| = 0$, then $z = x$. Furthermore, if i and $\|x\|$ are coprimes, and $x = z$, then $x = 0$ or $x = 2^{\|x\|} - 1$.

Factorizations. The more constraints are merged or reduced to simpler constraints, the closer we get to a proof. Functional factorization allows to detect instances based on equality of arguments, but some other instances can be factored as well, for example:

- from $x \oplus y = z$ and $x \oplus t = y$, we deduce that $t = z$, unifying the two variables and removing one of the constraints, now considered duplicates
- when $x \ll y_1 = z_1$ and $x \ll y_2 = z_2$ and $y_1 < y_2$, and z_1 is a constant, then one can infer the value of z_2. A similar operation can be carried out for $\gg$.

- two constraints $x \,\&\, y = 0$ and $x \mid y = 2^{\|x\|} - 1$ can be replaced by $x = \overline{y}$
- a constraint $x \,\&\, y = z$ (resp. $x \mid y = z$) is superfluous with the constraint $x = \overline{y}$ once z is deducted to be equal to 0 (resp. $2^{\|x\|} - 1$).
- the constraints $x = \overline{y}$ and $x \,\&\, z = y$ (resp. $x \mid z = y$) can both be removed once deducted that $x = 2^{\|x\|} - 1, y = z = 0$ (resp. $x = 0, y = z = 2^{\|x\|} - 1$).

4.4 BV-dedicated labeling strategies

A labeling strategy (a.k.a. search strategy) consists mainly of two heuristics: *variable selection* and *value selection*. For variable selection, we rely on the fail-first approach implemented in COLIBRI [33]. Basically, the variable to be selected is the one with the smallest domain (in terms of concretization). Adding the BV domain allows here to refine the notion of smallest. For value selection, in the event that BV is the smallest domain, our strategy is the following:

- First, we consider certain values that can simplify arithmetic constraints. In particular, we start by trying 0 (for $+, -, \times, /, \,\&\, , |, \oplus$), 1 (for $\times, /$) and $2^s - 1$ where s is the bitvector size (for $\,\&\, , |$);
- Second, we fix the value of several most significant and least significant unknown bits *(msub, lsub)* at once, allowing to strongly refine all domains thanks to inter-reduction, and to fix early the sign of the labeled variable (useful for signed BV operations). Currently, we fix at each labeling step one *msub* and two *lsub*, yielding 8 possible choices. We choose whether to set first or clear first in an arbitrary (but deterministic) way, using a fixed seed.

5 Experimentation

We describe in this section our implementation and experimental evaluation of CP(BV).

Implementation. We have implemented a BV support inside the COLIBRI CP solver [33] (cf. Section 3.3). Modular arithmetic domains and propagators are treated as blackbox, and we add the optimized bitlist domain and its associated propagators from [36], as well as all improvements discussed in Section 4. Building on top of COLBRI did allow us to prototype our approach very quickly, compared with starting from scratch.

Because it is written in a Prolog dialect, the software must be interpreted at each run, inducing a systematic 0.2 second starting time. This obstacle is not troubling for us because any real-world application would execute our software once and feed its queries in a chained manner through a server mode. Yet, the SMTCOMP rules impose that the software be called on command line with exactly one .smt2 file, which excludes a "server mode".

Experimental setup and caveats. We experiment CP(BV) on the 32k BV-formulas from the SMTCOMP benchmark, the leading competition of the SMT community. These formulas are mostly taken from verification-oriented industrial case-studies, and can be very large (up to several hundreds of MB). The first set of experiments (Section 5.1) has been run on the StarExec server[3] provided by SMTCOMP, they are made public [4]. The second set of experiments (Section 5.2) is run on a Intel® Core^{TM} i7-4712HQ CPU @ 2.30GHz with 16GB memory. Two points must be kept in mind.

- We fix a low time-out (60s) compared with the SMTCOMP rules (40 min), yet we argue that our results are still representative: first, such low time-outs are indeed very common in applications such as bug finding [25] or proof [19]; second, it is a common knowledge in first-order decision procedures that *"solvers either terminate instantly, or timeout"* – adding more time does not dramatically increase the number of solved formulas;
- SMT solvers such as Z3 and CVC4 are written in efficient compiled languages such as C/C++, with near-zero starting time. Hence, we have a constant-time disadvantage here – even if such a burden may not be so important in verification: since we are anyway attacking NP-hard problems, we are looking for exponential algorithmic improvements ; constant-time gains can only help marginally.

5.1 Evaluation against state-of-the-art benchmark

Absolute performance. Our implementation solved 24k formula out of 32k (75%). While it would not have permitted us to win the SMTCOMP, it is still a *significantly more thorough comparison with the SMT community than any previous CP effort on bitvectors*, demonstrating that CP can indeed be used for verification and bitvectors.

Comparing different choices of implementation. Improvements offered by our different optimizations are very dependent on the type of formulas, and these details would be diluted if regrouping all of the benchmarks. The reader is invited to consult our detailed results on the StarExec platform. Yet, as a rule of thumb, *our extensions yield a significant improvement on some families of examples, and do not incur any overhead on the other, proving their overall utility.* For example :

- On the family of formulas named `stp_samples`($\simeq$ 400 formulas), when deactivating the reductions from BV constraints to other domains (Section 4.2), the solver is unable to solve *a quarter less* formulas that it did with the full implementation. Removing the interreduction with the *Is* (Section 4.1), the loss rises to *half*;
- Solving the `spear` family suffers little from deactivating BV/*Is* inter-reductions, but *half* (200) formulas are lost without the BV constraints reducing other domains (Section 4.2);

[3] `www.starexec.org`

[4] `www.starexec.org/starexec/secure/explore/spaces.jsp?id=186070`

- On some other families, such as `pspace` and `dwp_formulas`, there is no tangible effect (neither positive nor negative) to the deactivation of improvements.

5.2 Comparison to state-of-the-art SMT solvers

We demonstrate in the following that CP(BV) is actually *complementary* to SMT approaches, especially on problems with large bitvectors or involving multi-theories. As competitors, we select the five best SMT solvers for BV theory: Z3, Yices, MathSAT, CVC4 and Boolector.

SMTCOMP: large formulas. To study the effect of our method on *scalability*, we show here the results on three categories of formulas, regrouped according to the (number of digits of the) size of the their largest bit-vectors: 3-digit (from 100 to 999), 4-digit and 5-digit, respectively having 629, 298 and 132 formulas. Results in Table 3 show on the one hand the *scalability* of CP(BV) – the larger BV sizes, the greater impact the CP approach has) – and on the other hand its *complementarity* with SMT. In particular, it shows the result of *duels* between CP(BV) and each of the SMT solvers : a formula is considered a win for a solver if it succeeds (TO = 60 seconds) while the other solver does not. We report results on a format Win/Lose (Solve), where Win and Lose are from CP(BV) point of view, and Solve indicates the number of formulas solved by the SMT solver. For example, MathSAT could solve 17 of the 132 5-digit formulas – all of which being solved by CP(BV), while CP(BV) could solve 63 formulas – 46 of which were unsolved by MathSAT. Here, *CP(BV) solves the higher number of 5-digit size formulas (equality with CVC4), and no solver but Boolector solves formulas that CP(BV) does not. On other sizes, CP(BV) solves less formulas, but it can still solve formulas that SMT solvers do not.*

Table 3: Comparing CP(BV) with five state-of-the-art solvers on large formulas

sz	#f	CP(BV) #solved	Z3 w/l (s)	Yices w/l (s)	MathSAT w/l (s)	CVC4 w/l (s)	Boolector w/l (s)
5	132	63	63/0 (0)	53/0 (10)	46/0 (17)	0/0 (63)	32/10 (41)
4	298	44	34/153 (163)	40/87 (91)	43/68 (69)	42/150 (152)	43/204 (205)
3	629	35	24/496 (507)	23/262 (274)	23/419 (431)	23/511 (523)	25/507 (517)

sz: size (#digits) - #f: # of formulas
w/l (s): #win/#lose for CP(BV), s: #formulas solved by SMT solver

SMTCOMP: hard formulas. We define *hard formulas* by separating 5 classes of difficulty, with class i regrouping the formulas on which i SMT solvers out of 5 spend more than 5 seconds. We compare CP(BV) to SMT solvers on these hard problems. Results are presented in Table 4, where we report for each class i the number of formulas from this class that CP(BV) solves quickly. Especially, the 5^{th} column (All-fail) shows that 61 formulas are solved only by CP(BV) in less than 5 seconds.

Mixing bitvectors and floats. Considering multi-theory formulas combining BV and FP arithmetics, COLIBRI has been tested on 7525 industrially-provided formulas (not

Table 4: Overall comparison on *hard examples*

Category	1-fail	2-fail	3-fail	4-fail	All-fail
#benchs	1083	382	338	1075	873
CP(BV) under 5s	139	108	10	68	61

publically available). It was able to solve 73% of them, standing half-way between Z3 / MathSAT and CVC4 (Table 5). Considering now the last SMTCOMP QF_BVFP category (Table 6), *even with the 0.2 seconds starting time, CP(BV) would have won the competition* – admittedly, there are only few formulas in this category.

Table 5: Industrial formula with bitvectors and floats

	CP(BV)	Z3	MathSAT	CVC4
#solved	5512	7225	7248	2245
ratio	73%	96%	96%	29%

Total: 7525 formulas

Table 6: SMTCOMP, QF_BVFP category

	Z3	MathSAT	CP(BV)
int_to_float_complex_2.smt2	1.04	0.13	0.25
int_to_float_simple_2.smt2	2.17	0.22	0.21
int_to_float_complex_1.smt2	0.95	0.08	0.25
int_to_float_simple_1.smt2	0.02	0.02	0.25
nan_1.smt2	0	0	0.26
incr_by_const.smt2	8.20	30.50	0.26
int_to_float_complex_3.smt2	1.89	0.44	0.25
quake3_1.smt2	TO	TO	TO

6 Related work

CP-based methods for BV. This work strongly stands upon the prior results of Bardin *et al.* [1] and Michel *et al.* [36]. Our respective contribution is already discussed at length in Sections 2 and 3. Basically, while we reuse the same general ideas, we sharpen them through a careful selection of the best aspects of each of these works and the design of new mechanisms, especially in terms of domain combination and labeling strategies. As a result, experiments in Section 5 demonstrate that our own CP(BV) approach performs much better than previous attempts. Moreover, we perform an extensive comparison with SMT solvers on the whole SMTCOMP benchmark, while these previous efforts were either limited to conjunctive formulas or remain only theoretical. The results by Michel *et al.* have been applied to a certain extent [46] as an extension of MiniSat [21], yet with no reduced product, to a limited set of BV operations and on BV sizes no larger than 64 bits. Older word-level approaches consider straightforward translations of

bit-vector problems into disjunctive or non-linear arithmetic problems [10, 22, 40, 45, 48, 49] (including bitblasting-like transformation for logical bitwise operators), and then rely on standard methods from linear integer programming or CP. Experimental results reported in [44, 1] demonstrate that such straightforward word-level encoding yield only very poor results on formulas coming from software verification problems.

SMT-based methods for BV. While state-of-the-art methods heavily rely on bit-blasting and modern DPLL-style SAT solvers [38, 43], the community is sensing the need for levels of abstraction "where structural information is not blasted to bits"[12]. Part of that need comes from the knowledge that certain areas, arithmetic for example, are not efficiently handled by bit-level reasoning tools. As a mitigation, SMT solvers typically complement optimized bitblasting [12, 13, 32] with word-level preprocessing [3, 24]. Compared to these approaches, we lack the highly-efficient learning mechanisms from DPLL. Yet, our domains and propagations yield more advanced simplifications, deeply nested with the search mechanism.

7 Conclusion

This work addresses the challenge of developing efficient Constraint Programming-based approaches for solving formulas over (the quantifier-free fragment of) the theory of bitvectors, which is of paramount importance in software verification. While the Formal Verification community relies essentially on the paradigm of SMT solving and reduction to Boolean satisfiability, we explore an alternative, high-level resolution technique through dedicated CP principles. We build on a few such anterior results and sharpen them in order to propose a highly efficient CP(BV) resolution method. We show that CP(BV) is much more efficient than the previous attempts from the CP community and that it is indeed able to solve the majority of the standard verification benchmarks for bitvectors. Moreover CP(BV) already complements the standard SMT approach on several crucial (and industry-relevant) aspects, such as scalability w.r.t. bit-width, formulas combining bitvectors with bounded integers or floating-point arithmetic, and formulas deeply combining non-linear arithmetic and bitwise operators.

Considering that our current CP(BV) approach is far from optimal compared with existing SMT solvers, we believe this work to be an important landmark toward building competitive CP-based verification-oriented solvers. Moreover, our approach clearly challenges the well-accepted belief that bitvector solving is better done through bitblasting, opening the way for a new generation of word-level solvers.

References

[1] S. Bardin, P. Herrmann, and F. Perroud. "An Alternative to SAT-Based Approaches for Bit-Vectors". In: *TACAS*. 2010.

[2] S. Bardin and P. Herrmann. "OSMOSE: Automatic Structural Testing of Executables". In: *Softw. Test. Verif. Reliab.* 21.1 (2011).

[3] C. Barret, D. Dill, and J. Levitt. "A decision procedure for bit-vector arithmetic". In: *DAC 98*. 1998.

[4] C. Barrett et al. "CVC4". In: *CAV*. 2011.

[5] C. Barrett et al. *The SMT-LIB Standard: Version 2.0*. Tech. rep. 2010.

[6] C. W. Barrett et al. "Satisfiability Modulo Theories". In: *Handbook of Satisfiability*. 2009.

[7] A. Biere et al. "Symbolic Model Checking without BDDs". In: *Tools and Algorithms for the Construction and Analysis of Systems*. 1999.

[8] N. Bjørner. "Taking Satisfiability to the Next Level with Z3". In: *Proceedings of the 6th International Joint Conference on Automated Reasoning*. 2012.

[9] B. Blanc et al. "Handling State-Machines Specifications with GATeL". In: *Electr. Notes Theor. Comput. Sci.* 264.3 (2010).

[10] R. Brinkmann and R. Drechsler. "RTL-datapath verification using integer linear programming". In: *15th Int. Conf. on VLSI Design*. 2002.

[11] R. Brummayer and A. Biere. "Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays". In: *TACAS*. 2009.

[12] R. Bruttomesso et al. "A Lazy and Layered SMT(BV) Solver for Hard Industrial Verification Problems". In: *Computer Aided Verification (CAV)*. 2007.

[13] R. E. Bryant et al. "Deciding Bit-Vector Arithmetic with Abstraction". In: *Tools and Algorithms for the Construction and Analysis of Systems TACAS*. 2007.

[14] A. Cimatti et al. "The MathSAT5 SMT Solver". In: *TACAS*. 2013.

[15] E. M. Clarke, D. Kroening, and F. Lerda. "A Tool for Checking ANSI-C Programs". In: *TACAS*. 2004.

[16] H. Collavizza, M. Rueher, and P. Hentenryck. "CPBPV: A Constraint-Programming Framework for Bounded Program Verification". In: *CP2008*. 2008.

[17] R. David et al. "BINSEC/SE: A Dynamic Symbolic Execution Toolkit for Binary-Level Analysis". In: *SANER 2016*. 2016.

[18] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., 2003.

[19] E. W. Dijkstra. *A discipline of programming*. Vol. 1. Prentice-Hall Englewood Cliffs, 1976.

[20] B. Dutertre. "Yices 2.2". In: *Computer-Aided Verification (CAV)*. Vol. 8559. 2014.

[21] N. Eén and N. Sörensson. "An Extensible SAT-solver". In: *Theory and Applications of Satisfiability Testing (SAT)*. 2003.

[22] F. Ferrandi, M. Rendine, and D. Sciuto. "Functional verification for SystemC descriptions using constraint solving". In: *Design, Automation and Test in Europe*. 2002.

[23] T. Feydy, A. Schutt, and P. J. Stuckey. "Global Difference Constraint Propagation for Finite Domain Solvers". In: *PPDP*. 2008.

[24] V. Ganesh and D. L. Dill. "A Decision Procedure for Bit-Vectors and Arrays". In: *CAV 2007*. 2007.

[25] P. Godefroid. "Test Generation Using Symbolic Execution". In: *FSTTCS*. 2012.

[26] A. Gotlieb. "TCAS software verification using Constraint Programming". In: *Knowledge Engineering Review* 27.3 (2012).

[27] A. Gotlieb, B. Botella, and M. Rueher. "Automatic Test Data Generation Using Constraint Solving Techniques". In: *ISSTA*. 1998.

[28] A. Gotlieb, M. Leconte, and B. Marre. "Constraint Solving on Modular Integers". In: 2010.

[29] T. A. Henzinger et al. "Lazy abstraction". In: *POPL*. 2002.

[30] D. Kroening and O. Strichman. *Decision Procedures: An Algorithmic Point of View*. 1st ed. Springer Publishing Company, Incorporated, 2008.

[31] M. Leconte and B. Berstel. "Extending a CP Solver with Congruences as Domains for Program Verification". In: *Trends in Constraint Programming*. 2010.

[32] P. Manolios and D. Vroon. "Efficient circuit to CNF conversion". In: *SAT 2007*.

[33] B. Marre and B. Blanc. "Test Selection Strategies for Lustre Descriptions in GaTeL". In: *ENTCS*. Vol. 111. 2005.

[34] B. Marre and C. Michel. "Improving the Floating Point Addition and Subtraction Constraints". In: *Principles and Practice of Constraint Programming - CP 2010*. 2010.

[35] K. L. McMillan. "Lazy Abstraction with Interpolants". In: *CAV*. 2006.

[36] L. D. Michel and P. Van Hentenryck. "Constraint Satisfaction over Bit-Vectors". English. In: *Principles and Practice of Constraint Programming*. Vol. 7514. 2012.

[37] R. A. D. Millo, R. J. Lipton, and A. J. Perlis. "Social Processes and Proofs of Theorems and Programs". In: *Communications of the Association of Computing Machinery* 22.5 (1979).

[38] M. W. Moskewicz et al. "Chaff: Engineering an Efficient SAT Solver". In: *Design Automation Conference, DAC*. 2001.

[39] G. Nelson and D. C. Oppen. "Simplification by Cooperating Decision Procedures". In: *ACM Trans. Program. Lang. Syst.* 1.2 (1979).

[40] G. Parthasarathy et al. "An efficient finite-domain constraint solver for circuits". In: *41th Design Automation Conf.* 2004.

[41] M. Pelleau et al. "A Constraint Solver Based on Abstract Domains". In: *VMCAI 2013*. 2013.

[42] J. D. Scott, P. Flener, and J. Pearson. "Bounded Strings for Constraint Programming". In: *ICTAI*. 2013.

[43] J. P. M. Silva and K. A. Sakallah. "GRASP: A Search Algorithm for Propositional Satisfiability". In: *IEEE Trans. Computers* 48.5 (1999).

[44] A. Sülflow et al. "Evaluation of SAT like proof techniques for formal verification of word level circuits". In: *8th IEEE Workshop on RTL and High Level Testing*. 2007.

[45] R. Vemuri and R. Kalyanaraman. "Generation of design verification tests from behavioral VHDL programs using path enumeration and constraint programming". In: *IEEE Transactions on VLSI Systems, 3(2), pp. 201-214*. 1995.

[46] W. Wang, H. Søndergaard, and P. J. Stuckey. "A Bit-Vector Solver with Word-Level Propagation". In: *CPAIOR*. 2016.

[47] N. Williams, B. Marre, and P. Mouy. "On-the-Fly Generation of K-Path Tests for C Functions". In: *ASE 2004*. 2004.

[48] Z. Zeng, M. Ciesielski, and B. Rouzeyre. "Functional test generation using Constraint Logic Programming". In: *11th Int. Conf. on Very Large Scale Integration of Systems-on-Chip*. 2001.

[49] Z. Zeng, P. Kalla, and M. Ciesielski. "LPSAT: a unified approach to RTL satisfiability". In: *4th Conf. on Design, Automation and Test in Europe*. 2001.

Une simple heuristique pour rapprocher DFS et LNS pour les COP

Julien Vion **Sylvain Piechowiak**

Université de Valenciennes et du Hainaut Cambrésis
LAMIH CNRS UMR 8201
{julien.vion, sylvain.piechowiak}@univ-valenciennes.fr

Résumé

Dans cet article, nous montrons comment une combinaison de stratégies de branchement et de redémarrages pour la recherche en profondeur d'abord (DFS) permet de reproduire le fonctionnement de la recherche par grand voisinage (LNS) pour la résolution de problèmes d'optimisation à contraintes, ce qui permet de rapprocher considérablement les deux techniques. En particulier, nous pouvons implémenter une stratégie DFS qui bénéficie des propriétés de passage à l'échelle de LNS tout en étant capable de prouver l'optimalité des solutions.

Abstract

In this paper, we argue that a combination of well-known and new search strategies enables Depth First tree Search (DFS) to mimic the behavior of Large Neighborhood Search (LNS), which reduces considerably the gap between the two techniques. In particular, we are able to implement a DFS strategy that shares the scalability properties of LNS, but the optimality of solutions can eventually be proven.

1 Introduction

Les stratégies de recherche utilisées pour résoudre les problèmes d'optimisation à contraintes (COP pour *Constraint Optimization Problem*) peuvent être complètes ou incomplètes. Les méthodes complètes sont généralement basées sur la recherche arborescente en profondeur d'abord (DFS pour *Depth-First Search*), guidée par une heuristique de branchement comme $^{\mathrm{dom}}/_{\mathrm{wdeg}}$ [2], couplée avec des techniques de filtrage comme la consistance d'arc ou encore la consistance aux bornes.

La recherche par grand voisinage (LNS pour *Large Neighborhood Search*) est une stratégie de recherche hybride [16] qui réalise des « déplacements » itératifs de manière similaire à une recherche locale, mais utilise une DFS et la propagation de contraintes pour améliorer la meilleure solution connue [17]. L'idée de LNS est de *relâcher* la meilleure solution connue en restaurant le domaine d'une partie de ses variables. Le « *fragment* » obtenu est alors *réoptimisé* par une DFS. Comme la plupart des stratégies incomplètes, LNS passe bien mieux à l'échelle qu'une DFS classique, mais elle ne peut pas prouver l'optimalité d'une solution (ou l'inconsistance d'un problème). De plus le choix des fragments à réoptimiser est une tâche difficile et dépendante du problème à traiter. Seuls quelques rares travaux ont été réalisés par le passé pour concevoir des heuristiques générales de sélection de voisinage (e. g., [8, 10, 13]), et peu d'expérimentations ont été menées sur de grandes variétés de problèmes.

Dans cet article, nous présentons une combinaison de stratégies de recherche : certaines, bien connues, sont décrites dans la section 2. D'autres, nouvelles, sont décrites dans la section 4. Ces stratégies permettent à DFS de se comporter de manière très similaire à LNS, comme nous l'expliquons en section 3. L'objectif est à la fois d'améliorer le passage à l'échelle de DFS sur des problèmes d'optimisation de grande taille, et d'éviter les principaux inconvénients de LNS que nous venons de décrire. Des expérimentations sur le problème de *Steel Mill Slab* ainsi que sur les instances du challenge MiniZinc 2016 sont présentées en section 5.

2 Éléments sur DFS

Un modèle d'optimisation discrète à contraintes est défini par un ensemble de n variables, chacune pouvant être instanciée à l'une des d valeurs d'un ensemble

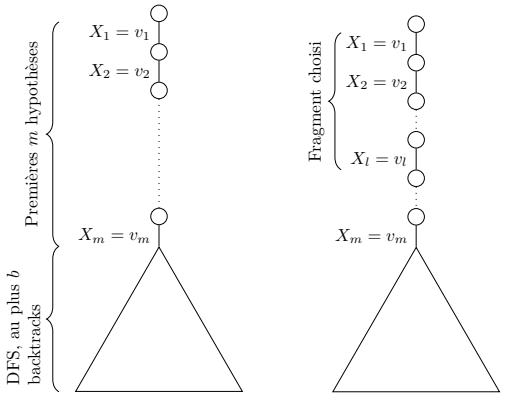
non-vide donné, ainsi qu'un ensemble de contraintes. Chacune d'elles définit les instanciations autorisées d'un sous-ensemble des variables du problème. Le problème d'optimisation discrète à contraintes (COP pour *Constraint Optimization Problem*) consiste à trouver une solution, c'est-à-dire une instantiation de toutes les variables du problème, qui maximise ou minimise la valeur d'une variable particulière, dite « de *coût* ». Ce problème est NP-difficile en général.

Une DFS combinée à la propagation des contraintes et une bonne heuristique d'affectation des variables est la stratégie la plus populaire pour résoudre les problèmes de satisfaction et d'optimisation de contraintes. La propagation est considérée comme le principal atout de la programmation par contraintes. Cette technique est particulièrement efficace en environnement « boite noire », c'est-à-dire n'autorisant pas une adaptation des algorithmes à un problème spécifique.

Pour résoudre une instance de COP, on se ramène généralement à une série de problèmes de décision. On commence par rechercher une solution au modèle, puis on réduit le domaine de la variable de coût pour engager la recherche d'une *meilleure* solution. On s'arrête quand on peut prouver qu'il n'existe plus de meilleure solution. La dernière solution est optimale.

On peut implémenter une DFS basée sur un arbre binaire. Lorsque la phase de propagation est terminée, l'espace de recherche restant est divisé en deux parties en appliquant soit une hypothèse logique, soit sa négation [6]. Cette hypothèse est choisie par une heuristique notée H_{DFS}. Le plus souvent, elle consiste à affecter une valeur à une variable. Si aucune solution n'est trouvée, on supprime la valeur du domaine de la variable. Le choix de la variable à instancier est extrêmement important, et a un impact énorme sur la taille de l'arbre de recherche. Une des heuristiques les plus efficaces est $^{\text{dom}}/_{\text{wdeg}}$ [2], bien que des stratégies inspirées par les solveurs SAT CDCL [11] semblent dominer les challenges MiniZinc depuis quelques années [12, 20]. Le choix de la valeur à affecter, cependant, a toujours été considéré comme de bien moindre impact. Il y a peu de littérature sur le sujet et les quelques tentatives semblent avoir eu des succès mitigés [7].

La principale faiblesse de DFS est son manque de flexibilité. L'arbre de recherche a une taille potentiellement exponentielle, et les hypothèses de branchement réalisées en haut de l'arbre ont peu de chances d'être reconsidérées. Une DFS binaire permet de changer de variable à affecter après un backtrack. Ceci permet restreindre les mauvais choix heuristiques en début de recherche. Cependant, la technique de diversification la plus populaire consiste à *redémarrer* la recherche de manière périodique, à condition que les heuristiques ne sélectionnent pas systématiquement les variables et



(a) DFS avec une limite de backtracks (b) LNS

FIGURE 1 – DFS *vs* LNS

valeurs dans le même ordre [5]. $^{\text{dom}}/_{\text{wdeg}}$ fait partie des heuristiques dites *adaptatives*, qui « apprennent » la structure du problème au cours de la recherche en fonction des conflits rencontrés. Les redémarrages permettent d'exploiter au plus vite l'information apprise. Ajouter une part d'aléatoire dans les heuristiques peut également permettre une diversification.

Les redémarrages peuvent être déclenchés selon des critères très différents. Le plus simple est de fixer une limite du nombre de backtracks à b. On augmente graduellement cette limite après chaque redémarrage, par exemple suivant une progression géométrique [22]. Ainsi, la limite peut devenir supérieure au nombre de backtracks nécessaires pour résoudre le problème, ce qui rend la recherche complète. Cependant, une progression géométrique tend à augmenter b très rapidement, ce qui limite le nombre global de redémarrages à un nombre logarithmique en fonction du nombre total de backtracks. Une stratégie plus aggressive se base sur la série Luby [9] de type 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8... Cette stratégie est également complète, et le nombre de redémarrages est linéaire par rapport au nombre total de backtracks. La communauté SAT a récemment développé une littérature autour de stratégies dynamiques, plus sophistiquées, influencées par les techniques d'apprentissage de clauses qui sont encore mal établies en CP [1].

3 Rapprocher DFS et LNS

Avant de décrire notre contribution, nous faisons une observation décrite par la fig. 1 : la forme d'un arbre de recherche obtenu en utilisant DFS avec redémarrages (i. e., avec une limite du nombre de back-

tracks b) est très similaire à une itération de LNS. Notons X_i (resp. v_i) la $i^{\text{ème}}$ variable (resp. valeur) choisie par l'heuristique de choix de variable (resp. valeur). Nous définissons m comme le dernier niveau de l'arbre de recherche où aucun backtrack n'a eu lieu, i. e., l'hypothèse $X_m = v_m$ a été testée mais sa consistance n'a pu être décidée en raison de la limite du nombre de backtracks. Si et seulement si $m = 0$, l'ensemble de l'espace de recherche a été exploré.

Pour DFS (fig. 1a), les m premières instanciations sont choisies suivant l'heuristique H_{DFS}, puis un arbre de recherche d'au plus b backtracks est exploré. La recherche est interrompue si une solution est trouvée ou si la limite de backtracks est atteinte. Il y a un lien très clair entre m et b, de l'ordre de $b \in O(d^{n-m})$: plus b est grand, plus m sera faible et *vice-versa*.

Pour LNS (fig. 1b), les variables X_1 à X_l constituent le « fragment » à explorer et les valeurs correspondant à la meilleure solution connue leurs sont affectées. La taille et les variables du fragment sont choisies par une heuristique H_{LNS}. Puis une DFS est exécutée sur le reste de l'espace de recherche, en utilisant H_{DFS}. On fixe également une limite b pour éviter de bloquer sur un sous-problème trop difficile et pour permettre de diversifier la recherche même si l est trop faible. La recherche est interrompue si une solution est trouvée, si le nombre maximal de backtracks fixé est atteint ou si le sous-problème est prouvé inconsistant. Par définition de l'algorithme LNS, on a $m \geq l$. On aura $m = l$ si et seulement si le sous-problème est prouvé inconsistant, ce qui nécessite $b \in O(d^{n-l})$. On peut dégénérer LNS en un DFS si $l = 0$; dans le cas contraire on aura $m > 0$: LNS est un algorithme incomplet.

La principale différence entre les deux stratégies est qu'avec LNS, les l premières hypothèses consistent à instancier le fragment choisi par H_{LNS} avec les valeurs de la meilleure solution connue, alors que pour DFS la stratégie de branchement est *a priori* homogène[1]. Notons également qu'il est possible que le fragment choisi par LNS soit immédiatement détecté comme inconsistant par propagation. Une variante nommée PGLNS pour *Propagation-Guided* LNS instancie et propage chaque variable du fragment de manière itérative pour restreindre ce cas de figure, et exploiter l'impact de la propagation pour construire les fragments [13]. Cette dernière stratégie rapproche plus encore LNS de DFS. Concernant les stratégies de redémarrage pour LNS, il est habituel de la faire évaluer en fonction de la facilité à trouver une meilleure solution (i. e., b est réduit quand une solution est trouvée, augmenté sinon). Le résultat est alors proche de ce qui est fait en DFS (cf section 2).

Dans la section suivante, nous proposons de construire une stratégie de DFS en exploitant la principale caractéristique de LNS : affecter les valeurs de la meilleure solution connue aux variables. On peut fusionner les heuristiques H_{DFS} et H_{LNS}, et en abandonner la notion de fragment de taille l : on obtient alors une stratégie très proche de LNS à travers une simple heuristique de choix de valeur.

4 L'heuristique de choix de valeur *Best-Solution*

Nous proposons une heuristique de choix de valeur, que nous nommons *Best-Solution* (BS). Combinée à une heuristique de choix de variable quelconque notée H_{var}, elle définit une heuristique de branchement pour DFS. BS requiert également une heuristique de choix de valeurs « de repli » quelconque notée H_{val}.

Soit S la meilleure solution connue du COP, s'il y en a une. Si S est disponible, on note S_i la valeur correspondant à la variable X_i. Si X_i est la variable choisie par H_{var}, alors BS s'opère en deux étapes :

1. Si S est disponible et S_i appartient au domaine courant de X_i, alors choisir S_i.
2. Sinon, choisir une valeur du domaine de X_i selon l'heuristique H_{val}.

Cette stratégie est très simple et très adaptable : on peut utiliser une heuristique connue pour H_{var}, H_{val}, ou concevoir de nouvelles heuristiques adaptées au problème à résoudre. On peut également manipuler H_{var} pour exploiter les heuristiques de choix de fragment couramment utilisées par LNS. La stratégie utilisée pour les restarts est tout aussi libre. Cependant, pour obtenir un comportement plus proche d'une recherche locale, notamment au niveau des caractéristiques de passage à l'échelle, nous pensons qu'il est préférable d'utiliser une politique de redémarrage aggressive, et de biaiser les heuristiques H_{var} et H_{val} pour augmenter la diversification de la recherche.

Pour H_{var}, nous suggérons d'utiliser une heuristique connue et efficace, si possible adaptative, permettant la diversification. Nous avons opté pour $^{\text{dom}}/_{\text{wdeg}}$ dans nos expérimentations, mais il est envisageable de concevoir des heuristiques inspirées par LNS comme *Propagation-Guided* [13] ou *Cost-Impact* [10]. Pour améliorer la diversification, nous suggérons d'utiliser au minimum un départage aléatoire en cas d'égalité. On peut augmenter encore la diversification : nous fixons une probabilité p avec laquelle la variable à affecter sera choisie aléatoirement, au lieu de faire confiance

1. Il reste possible de construire H_{DFS} de sorte que les variables soient choisies selon une politique différente pour les premières variables à affecter, qui pourraient correspondre aux l variables du fragment LNS.

à l'heuristique. Nous appelons cette stratégie « diversification aléatoire ». Si $p = 0$, l'heuristique obtenue est strictement équivalente à H_{var} ; si $p = 1$, le comportement est complètement aléatoire.

En ce qui concerne H_{val}, il y a peu de littérature sur le sujet, et, comme dans la plupart des cas, il est relativement efficace de choisir la borne inférieure du domaine de la variable, c'est-à-dire affecter les valeurs par ordre lexicographique. Cette heuristique est souvent plus efficace qu'une stratégie purement aléatoire car elle correspond à des propriétés des problèmes et des algorithmes de propagation. Une technique populaire issue de SAT, qui pourrait d'ailleurs sembler similaire à notre proposition, est le *Phase Saving* [14]. Elle consiste à choisir en priorité la même valeur que la dernière affectée pour les variables. Cette stratégie ne peut avoir d'impact qu'après un redémarrage, un backjump, ou encore un changement de variable à affecter dans un branchement binaire : après un retour-arrière simple, la dernière valeur affectée n'est généralement plus disponible. L'objectif est de réutiliser ainsi les solutions des sous-problèmes difficiles situés en haut de l'arbre de recherche. Ce n'est pas nécessaire pour SAT, mais une implémentation pour des variables non booléennes nécessite également une heuristique de repli au cas où la dernière valeur affectée ne serait plus disponible. Nos expérimentations en section 5 montrent que BS est plus robuste que le Phase Saving sur les benchmarks évalués.

5 Expérimentations

Nous avons utilisé notre solveur *Concrete 3.3* [21] sur un ensemble de machines équipées de CPU Intel Core i5-6500 @ 3,2 GHz. Notre solveur est implémenté à l'aide du langage et de l'API Scala 2.12.1 et fonctionne sur une JRE Java 8 update 121 Server avec 4 Gio de mémoire de tas autorisée.

5.1 Environnement « boite noire » : le challenge MiniZinc

Évaluer les performances d'algorithmes d'optimisation n'est pas trivial. Trouver la solution optimale est dans la plupart des cas hors de la portée des solveurs : l'objectif est plutôt de trouver la meilleure solution possible dans le temps imparti. Cependant, la qualité des solutions n'est pas comparable d'une instance à l'autre. Nous avons choisi d'utiliser ici le même système d'évaluation que celui conçu pour le challenge MiniZinc, basé sur le système de vote de Borda [18] : « Chaque instance est considérée comme un votant qui trie les solveurs en fonction de la qualité du résultat obtenu. Pour chaque instance, chaque solveur s gagne des points en comparant son résultat par rapport à chaque autre solveur s' sur cette même instance :

— Si s obtient une meilleure solution que s', i. e., la variable de coût est plus proche de l'optimal, ou s prouve l'optimalité alors que s' échoue à le faire dans une limite de 1 000 secondes, il obtient 1 point.

— Si s et s' (avec des temps d'exécution respectifs t et t') donnent un résultat identique, le score obtenu se base sur une comparaison des temps d'exécution $\frac{t'}{t+t'}$, à une exception près :

— Si s donne une solution moins bonne que s', ou encore ne trouve aucune solution sans prouver l'inconsistance de l'instance, il obtient 0 point, même si s' échoue également. »

Le solveur ayant le meilleur score cumulé remporte le challenge. Nous comparons l'efficacité de notre solveur paramétré avec chaque combinaison des stratégies suivantes :

Choix de variable : $^{\text{dom}}/_{\text{wdeg}}$ avec diversification aléatoire, $p = 0$ et 20 %.

Choix de valeur : Lexicographique (Lex), Phase Saving (PS) ou Best-Solution (BS). PS et BS utilisent Lex comme heuristique de repli si la valeur sélectionnée n'est plus disponible.

Strategy de redémarrage : Geometrique (base 100, croissance 20 %) ou Luby (base 100).

Bien sûr, nous voudrions expérimenter plus de stratégies et de combinaisons, certains choix pouvant paraitre arbitraires. Cependant, le nombre de combinaisons croît exponentiellement, les interactions entre les paramètres sont complexes et il est difficile de les analyser séparément. Notons que le système de Borda n'a pas la propriété d'*indépendance des alternatives non-pertinentes*, c'est-à-dire qu'ajouter ou supprimer un paramétrage peut changer le classement des autres solveurs. Nous avons été contraints de réduire le nombre de paramètres pour conserver une certaine lisibilité. Nous pensons que la liste ci-dessus est représentative. De manière un peu surprenante, des résultats préliminaires ont montré que la valeur du paramètre p n'avait que peu d'influence sur les performances, tant qu'il reste compris entre 10 et 50 %.

Nous avons utilisé les 100 benchmarks (5 instances pour chacun des 20 problèmes proposés) du challenge MiniZinc 2016. La principale motivation de ce travail était clairement d'améliorer les résultats du solveur *Concrete* dans les conditions du challenge MiniZinc. *Concrete 3.2* s'était classé 19e des 22 solveurs en compétition en 2016, en utilisant un paramétrage classique, i. e., $^{\text{dom}}/_{\text{wdeg}}$, $p = 0$, Lex et redémarrages

TABLE 1 – Résultats sur les instances du MiniZinc Challenge 2016 (MZNC) et le problème Steel Mill Slab. Pour chaque catégorie, nous avons mis en valeur les trois meilleurs scores (si un score est inférieur de moins de 5 % au 3e meilleur score, il est également mis en valeur).

H_{var}	dom/wdeg, $p = 0$						dom/wdeg, $p = 20\,\%$					
Redémarrages	Geometric			Luby			Geometric			Luby		
H_{val}	Lex	PS	BS	Lex	PS	BS	Lex	PS	BS	Lex	PS	BS
carpet-cutting	9	6	9	7	7	7	37	38	**41**	**48**	**44**	**47**
celar	3	29	35	12	27	31	9	29	**52**	21	**37**	**45**
cryptanalysis*	8	0	8	**16**	0	**16**	9	0	9	**16**	0	**16**
depot-placement	17	**42**	32	8	**39**	29	19	**42**	34	13	**38**	17
diameterc-mst	0	0	0	0	0	0	7	**27**	10	**26**	**29**	**33**
elitserien	0	0	0	0	0	0	0	0	0	0	0	0
filters	23	23	19	23	30	19	27	28	28	**39**	**38**	**34**
gbac	7	29	25	5	25	22	6	**47**	**33**	6	**49**	31
gfd-schedule	20	24	26	**37**	16	26	33	17	**39**	29	13	**39**
java-auto-gen	24	22	26	**31**	16	19	26	11	**36**	**32**	**34**	**32**
mapping	29	**32**	**30**	**33**	24	28	**31**	11	**30**	**31**	26	24
maximum-dag	30	26	30	26	17	22	**35**	23	**40**	24	21	**36**
mrcpsp	18	12	14	13	7	6	31	**33**	30	**33**	30	**38**
nfc	25	25	20	26	26	19	**31**	**31**	**31**	**32**	**32**	**32**
oocsp-racks*	6	13	5	5	5	5	15	16	15	**23**	**25**	**22**
prize-collecting	28	6	31	32	13	**50**	26	3	**38**	35	8	**53**
rcpsp-wet	16	20	28	15	18	24	16	**43**	41	21	**45**	**44**
solbat*	**31**	**31**	**32**	22	14	22	28	20	27	12	10	12
tpp	28	32	**42**	13	16	32	32	**38**	19	29	**39**	10
zephyrus	26	**36**	**38**	25	28	**38**	21	29	31	15	19	23
overall MZNC	350	411	451	346	324	413	439	486	**586**	484	**537**	**587**
steel-mill-slab	1907	528	**2500**	2227	582	**2540**	1890	615	**2720**	2254	671	**2884**

* : Problème de décision

géométriques. *Concrete 3.3* avec la meilleure combinaison testée ($p = 20\,\%$, BS et redémarrages Luby) se serait classé 15e. Les résultats sont présentés sur la table 1. Pour les problèmes de décision indiqués, BS est équivalent à Lex.

Avec BS et $p = 20\,\%$, les stratégies de redémarrage choisies font finalement peu de différence, mais avec des redémarrages Luby le solveur est plus souvent parmi les 3 meilleurs d'une instance donnée : ceux-ci seraient donc plus robustes.

5.2 Problème du Steel Mill Slab

Le problème du Steel Mill Slab Design Problem (CSPLib #38 [4]) est une variante de problème de Bin Packing : les objets sont *colorés* et au plus deux couleurs peuvent être placées dans une même boite (appelée ici *slab*) ; la taille de chaque slab peut être choisie parmi un ensemble donné, et l'objectif du problème est de minimiser la capacité inutilisée des slabs. Bien que le problème présente d'importantes symétries (les slabs sont interchangeables), celles-ci ne sont pas traitées efficacement avec les traditionnelles contraintes d'inégalité. Les premiers travaux publiés résolvant le Steel Mill Slab se basaient sur LNS avec une heuristique de choix de fragment aléatoire, et une taille de fragment fixe [3]. Nous avons ajouté deux contraintes au modèle pour supprimer certaines symétries : si un slab b est vide, alors le slab $b + 1$ doit être vide ; et deux objets identiques (taille et couleur) $i_1 < i_2$ doivent être placés dans des slabs $b_1 \leq b_2$. Nous avons également désactivé le départage aléatoire des égalités de dom/wdeg afin de départager en fonction du poids des objets. Des travaux plus récents ont montré qu'une résolution dynamique des symétries était bien plus efficace [19], mais il est impossible de modéliser de telles techniques avec le langage MiniZinc : il faut réaliser une implémentation spécifique au solveur.

Nous avons réalisé des expérimentation sur les 381 instances de SCHAUS et al. [15]. La table 1 montre que notre meilleur paramétrage, utilisant l'heuristique BS, les redémarrages Luby et la diversification aléatoire permet de résoudre les Steel Mill Slab bien plus efficacement qu'une DFS classique. Le Phase Saving est complètement inefficace sur ce problème.

6 Conclusion & perspectives

Dans cet article, nous avons montré qu'une DFS associée à une heuristique de choix de valeurs choisissant prioritairement les valeurs issues de la meilleure solution connue, une heuristique de choix de variables agrémentée d'une diversification aléatoire, et d'une stratégie de redémarrages améliore clairement le comportement d'une DFS classique dans un environnement « boite noire » en conservant, notamment, la complétude. Nous avons comparé le comportement de cette stratégie avec une LNS, ce qui a permis de rapprocher les deux méthodes de recherche.

Dans des travaux futurs, nous souhaitons développer la partie expérimentale afin de mieux analyser la proximité entre LNS et DFS+BS. Nous souhaitons également nous inspirer des travaux sur les heuristiques de choix de fragment en LNS pour dériver de nouvelles heuristiques de choix de variable pour DFS+BS. Ainsi, peut-être que cette nouvelle stratégie se révèlera aussi plus performante pour la résolution de problèmes industriels de très grande taille pour lesquels une approche purement complète reste inefficace.

Références

[1] A. Biere et A. Fröhlich. « Evaluating CDCL Restart Schemes ». In : *Pragmatics of SAT*. 2015.

[2] F. Boussemart, F. Hemery, C. Lecoutre et L. Saïs. « Boosting Systematic Search by Weighting Constraints ». In : *Proc. of the 16th European Conference on Artificial Intelligence (ECAI)*. 2004, p. 146–150.

[3] A. Gargani et P. Refalo. « An Efficient Model and Strategy for the Steel Mill Slab Design Problem ». In : *Proc. of the 13th Itl. Conf. on Principles and Practice of Constraint Programming (CP)*. Springer, sept. 2007, p. 77–89.

[4] I.P. Gent et T. Walsh. *CSPLib : a benchmark library for constraints*. Rapp. tech. APES-09-1999, 1999. url : http://www.csplib.org.

[5] C.P. Gomes, B. Selman, N. Crato et H. Kautz. « Heavy-tailed phenomena in satisfiability and constraint satisfaction problems ». In : *Journal of Automated Reasoning* 24 (2000), p. 67–100.

[6] J. Hwang et D.G. Mitchell. « 2-way vs d-way branching for CSP ». In : *Proc. of the 11th Itl. Conf. on Principles and Practice of Constraint Programming (CP)*. 2005, p. 343–357.

[7] C. Lecoutre, L. Saïs et J. Vion. « Using SAT Encodings to Derive CSP Value Ordering Heuristics ». In : *JSAT Special Issue on SAT/CP Integration* 1 (2007), p. 169–186.

[8] M. Lombardi et P. Schaus. « Cost Impact Guided LNS ». In : *Proc. of the 11th Itl. Conf. on Integration of AI and OR Techniques in CP (CPAIOR)*. Springer, 2014, p. 293–300.

[9] M. Luby, A. Sinclair et D. Zuckerman. « Optimal speedup of Las Vegas algorithms ». In : *Inform. Process. Lett.* 1993, p. 173–180.

[10] J.-B. Mairy, P. Schaus et Y. Deville. « Generic Adaptive Heuristics for Large Neighborhood Search ». In : *Proc. of the 7th Itl. Workshop on Local Search Techniques in Constraint Satisfaction (LSCS)*. 2010.

[11] J. P. Marques Silva et K. A. Sakallah. « GRASP – A new search algorithm for satisfiability ». In : *Proc. of Itl. Conf. on Computer Aided Design*. Nov. 1996, p. 220–227.

[12] O. Ohrimenko, P. J. Stuckey et M. Codish. « Propagation via Lazy Clause Generation ». In : *Constraints* 14.3 (sept. 2009), p. 357–391.

[13] L. Perron, P. Shaw et V. Furnon. « Propagation Guided Large Neighborhood Search ». In : *Proc. of the 10th Itl. Conf. on Principles and Practice of Constraint Programming (CP)*. 2004, p. 468–481.

[14] K. Pipatsrisawat et A. Darwiche. « A Lightweight Component Caching Scheme for Satisfiability Solvers ». In : *Proc. of the 10th Itl. Conf. on Theory and Applications of Satisfiability Testing (SAT)*. Springer, 2007, p. 294–299.

[15] P. Schaus, P. Van Hentenryck, J.-N. Monette, C. Coffrin, L. Michel et Y. Deville. « Solving Steel Mill Slab Problems with Constraint-Based Techniques : CP, LNS, and CBLS ». In : *Constraints* (2011).

[16] B. Selman, H. Kautz et D. McAllester. « Ten challenges in propositional reasoning and search ». In : *Proc. of the 15th Itl. Joint Conference on Artificial Intelligence (IJCAI)*. 1997.

[17] P. Shaw. « Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems ». In : *Proc. 4th Itl. Conf. on Principles and Practice of Constraint Programming (CP)*. 1998, p. 417–431.

[18] P. J. Stuckey, T. Feydy, A. Schutt, G. Tack et J. Fischer. « The MiniZinc Challenge 2008–2013 ». In : *AI Magazine* 35.2 (2014), p. 55–60.

[19] P. Van Hentenryck et L. Michel. « The Steel Mill Slab Design Problem Revisited ». In : *Proc. of the 5th Itl. Conf. on Integration of AI and OR Techniques in CP*. T. 5015. 2008, p. 377–381.

[20] M. Veksler et O. Strichman. « Learning General Constraints in CSP ». In : *Proc. 12th Itl Conf on Integration of AI and OR techniques in CP (CPAIOR)*. 2015.

[21] J. Vion. *Concrete : a CSP Solving API for the JVM*. `http://github.com/concrete-cp`. 2006–2016.

[22] T. Walsh. « Search in a small world ». In : *Proc. 16th Itl Joint Conf on Artificial Intelligence (IJCAI)*. 1999.

Une famille de règles d'élimination de variables pour les CSP binaires basées sur BTP

Achref El Mouelhi
Marseille - France
elmouelhi.achref@gmail.com

Résumé

L'étude des triangles cassés devient de plus en plus ambitieuse, par la résolution des problèmes de satisfaction de contraintes (CSP) en temps polynomial d'un coté, et par la réduction de l'espace de recherche à travers l'élimination de variables et la fusion de valeurs de l'autre. Pour cela, plusieurs extensions de ce concept ont été étudiées dans le passé récent, tel que les triangles cassés duaux et les triangles légèrement cassés. Ces extensions ont été introduites dans le but de maximiser soit le nombre de valeurs fusionnées et/ou le nombre d'instances traitables capturées. Mais, aucune d'entre elles n'a préservé toutes les caractéristiques de BTP.

Ici, nous introduisons une nouvelle version légère de BTP, que nous appelons m-fBTP (pour flexible broken-triangle property). m-fBTP permet la fusion de valeurs, l'élimination de variables et définit une plus grande classe polynomiale pour laquelle la cohérence d'arc est une procédure de décision.

Une version plus détaillée en langue anglaise a été publiée à AAAI'17 [4].

1 Définitions

Le problème de satisfaction de contraintes constitue un formalisme important pour la modélisation et la résolution des plusieurs problèmes du monde réel en Intelligence Artificielle. Ici, nous nous intéressons uniquement aux instances binaires. Une instance de CSP binaire est un couple $I = (X, C)$. $X = \{x_1, ..., x_n\}$ est un ensemble n **variables**. Chaque variable x_i a un ensemble fini de valeurs, appelé **domaine** et noté $D(x_i)$. C est un ensemble de e **contraintes** binaires. Chaque contrainte binaire C_{ij} est un couple $(Scp(C_{ij}), Rel(C_{ij}))$, où $Scp(C_{ij})$ est un ensemble de deux variables $\{x_i, x_j\}$ et $Rel(C_{ij})$ définit la compatibilité entre les valeurs dans $D(x_i)$ et $D(x_j)$.

Vérifier s'il existe une solution pour une instance de CSP binaire donnée est un problème NP-complet. Mais, en imposant quelques restrictions sur le domaine des variables ou sur les contraintes, une telle instance peut être résolue en temps polynomial. Par exemple, il est connu que les instances de CSP binaires qui satisfont BTP (pour Broken-Triangle Property [2]) sont résolues en temps polynomial par le niveau de cohérence le plus basique et le moins couteux en terme de complexité, à savoir la cohérence d'arc. BTP interdit l'existence d'un triangle cassé sur la variable courante vis-à-vis de variables précédentes dans l'ordre. Ici nous proposons une extension de BTP, appelée m-fBTP, qui préserve toutes ses caractéristiques.

Définition 1 *Un couple de valeurs $v'_k, v''_k \in D(x_k)$ satisfait **m-fBTP**, si pour chaque triangle cassé (v'_k, v_i, v_j, v''_k) avec $v_i \in D(x_i)$ et $v_j \in D(x_j)$, il existe un ensemble de $r \leq m$ **variables soutiens** $E \subseteq X \setminus \{x_i, x_j, x_k\}$ tel que pour toute solution partielle $\mathcal{A}$ de E, il existe $v_\ell \in \mathcal{A}$ tel que si $(v_\ell, v_i) \in Rel(C_{\ell i})$, alors $(v_\ell, v_j) \notin Rel(C_{\ell j})$. Dans ce cas, nous disons que (v'_k, v_i, v_j, v''_k) est un triangle cassé flexible. Une variable $x_k \in X$ satisfait m-fBTP si pour chaque couple de valeurs dans $D(x_k)$ satisfait m-fBTP.*

S'il n'existe aucun ensemble de variables pour soutenir un triangle cassé, alors ce dernier sera dit triangle purement cassé.

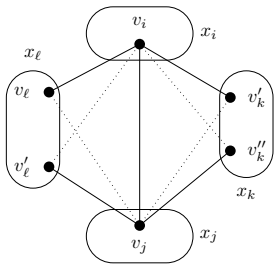


FIGURE 1 – La variable x_k satisfait m-fBTP.

La figure 1 montre un couple de valeurs $v'_k, v''_k \in$

$D(x_k)$ qui satisfait 1-fBTP car l'unique triangle cassé (v'_k, v_i, v_j, v''_k) est flexible (soutenu par x_ℓ).

Intuitivement, nous pouvons constater que si un couple de valeurs satisfait m-fBTP, alors il satisfait aussi $(m+1)$-fBTP.

2 Fusion de valeurs

Le résultat sur la fusion s'appuie sur la proposition suivante qui établit le lien avec m-wBTP [1].

Proposition 1 *Dans une instance de CSP binaire $I = (X, C)$, $\forall m,\ 0 \leq m \leq n-4$, si un couple de valeurs $v'_k, v''_k \in D(x_k)$ satisfait m-fBTP, alors il satisfait aussi m-wBTP.*

Puisque la fusion d'un couple de valeurs satisfaisant m-wBTP ne modifie pas la satisfiabilité d'une instance, alors nous pouvons aussi déduire que fusionner un couple de valeurs qui satisfait m-fBTP n'affecte pas la satisfiabilité d'une instance.

3 Élimination de variables

La fusion de valeurs d'une variable x_k qui satisfait m-wBTP ne peut se dérouler d'une façon itérative jusqu'à obtention d'une valeur singleton. En effet, après avoir fusionné un couple de valeurs $v'_k, v''_k \in D(x_k)$, il est possible que la variable x_k ne soit plus m-wBTP (voir section 5 de [1]). Ceci n'est pas le cas pour m-fBTP.

Lemme 1 *Étant donnée une variable x_k qui satisfait m-fBTP, après fusion d'un couple de valeurs $v''_k, v'''_k \in D(x_k)$ en une nouvelle valeur v'_k, aucun triangle purement cassé ne peut s'introduire sur x_k.*

Le théorème suivant est une conséquence directe des résultats précédents.

Théorème 1 *Étant donnée une instance de CSP binaire arc cohérente $I = (X, C)$, si une variable $x_k \in X$ satisfait m-fBTP, alors elle peut être éliminée de I tout en préservant la satisfiabilité.*

m-fBTP définit aussi une condition maximale d'élimination de variables, c'est-à-dire l'élimination de toute autre variable d'une instance I, sur laquelle il existe un triangle purement cassé, conduit nécessairement à la modification de la satisfiabilité I.

4 Classe polynomiale résolue par la cohérence d'arc

Nous étendons la définition de m-fBTP aux instances de CSP binaire.

Définition 2 *Une instance de CSP binaire I menée d'un ordre $<$ sur les variables satisfait m-fBTP par rapport à $<$ si pour toute variable x_k, chaque couple de valeurs dans $D(x_k)$ satisfait m-fBTP dans la sous-instance de I contenant les variables $x_i \leq x_k$.*

Comme BTP, m-fBTP est conservative, c'est-à-dire la propriété sera préservée même après application de tout algorithme de filtrage supprimant seulement des valeurs, comme la cohérence d'arc. Ce résultat est nécessaire pour prouver le théorème suivant.

Théorème 2 *La cohérence d'arc est une procédure de décision pour toute instance de CSP binaire $I = (X, C)$ qui satisfait m-fBTP ($1 \leq m \leq n-3$).*

m-fBTP constitue ainsi la plus large classe polynomiale résolue itérativement par l'élimination de variables.

5 Conclusion

Dans ce papier, nous avons introduit m-fBTP, une version légère de BTP, qui couvre les imperfections des versions précédentes et qui préserve toutes ses caractéristiques, fusion de valeurs, élimination de variables, conservation et résolution par la cohérence d'arc. Nous avons aussi prouver que m-fBTP es différente de k-BTP [3] et WBTP [5]. Il serait intéressant de valider expérimentalement ces résultats théoriques et les étendre aux instances CSP d'arité quelconque.

Références

[1] Martin C. Cooper, Achref El Mouelhi, and Cyril Terrioux. Extending broken triangles and enhanced value-merging. In *Proceedings of the 22nd International Conference on Principles and Practice of Constraint Programming (CP)*, pages 173–188, 2016.

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

[3] Martin C. Cooper, Philippe Jégou, and Cyril Terrioux. A microstructure-based family of tractable classes for CSPs. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP)*, pages 74–88, 2015.

[4] Achref El Mouelhi. A BTP-based family of variable elimination rules for binary CSPs. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 3871–3877, 2017.

[5] Wady Naanaa. Extending the broken triangle property tractable class of binary CSPs. In *Proceedings of the 9th Hellenic Conference on Artificial Intelligence (SETN)*, pages 3 :1–3 :6, 2016.

La contrainte globale `MinArborescence` pour les problèmes d'arborescence de poids minimum

V. R. Houndji[1,2]***P. Schaus**[2] **M. N. Hounkonnou**[1] **L. Wolsey**[2]

[1] Université d'Abomey-Calavi (UAC), Bénin
[2] Université catholique de Louvain (UCL), Belgique
ratheil.houndji@uac.bj

Résumé

Une arborescence recouvrante de poids minimum (ARPM) enracinée en un noeud r d'un graphe dirigé est un arbre couvrant dirigé de racine r et de poids minimum. L'article [4] introduit la contrainte globale `MinArborescence` pour les problèmes d'arborescence contraints (PAC) qui consistent à trouver une ARPM qui respecte d'autres contraintes supplémentaires. Le filtrage de cette contrainte peut se faire avec les coûts réduits de la programmation linéaire (PL) en $O(n^2)$ avec n le nombre de noeuds du graphe. Ce papier est un résumé de [4] dans lequel est également proposé un algorithme pour améliorer les coûts réduits de la PL. L'efficacité du filtrage basé sur les coûts réduits a été montrée sur une variante du PAC, l'ARPM avec des contraintes de ressource sur chaque noeud (ARPM-R).

1 Introduction

Une arborescence enracinée en r d'un graphe dirigé $G = (V, E)$ est un sous graphe sans cycle de G tel qu'il existe un chemin de r vers chacun des autres noeuds du graphe. Le poids d'une arborescence est la somme des poids de ses arcs. Le problème de l'arborescence recouvrante de poids minimum (ARPM) peut être formulé comme suit [3, 1] :

$$w(A(G)^\star) = \min \sum_{(i,j)\in E} w(i,j) \cdot x_{i,j} \tag{1}$$

$$\text{(ARPM)} \qquad \sum_{(i,j)\in\delta_j^{in}} x_{i,j} = 1, \forall j \in V \setminus \{r\} \tag{2}$$

$$\sum_{(i,j)\in\delta_S^{in}} x_{i,j} \geq 1, \forall S \subseteq V \setminus \{r\} : |S| \geq 2 \tag{3}$$

$$x_{i,j} \in \{0,1\}, \forall (i,j) \in E \tag{4}$$

dans lequel $x_{i,j} = 1$ si l'arc (i,j) est dans l'ARPM $A(G)^\star$ et $x_{i,j} = 0$ sinon ; δ_S^{in} est l'ensemble des arcs entrant dans S : $\delta_S^{in} = \{(i,j) \in E : (i \in V \setminus S) \wedge (j \in S)\}$; $w(i,j)$ est le poids de l'arc (i,j). Le premier groupe de contraintes impose qu'il y ait exactement un arc qui entre dans chaque noeud $j \in V \setminus \{r\}$ et les contraintes (3) imposent l'existence d'un chemin à partir de la racine r vers chacun des autres noeuds. Sans perte de généralité [2], nous supposons que $w(i,i) = \infty, \forall i \in V$ et $w(i,j) > 0, \forall (i,j) \in E$.

En considérant le graphe de la figure 1, le sous-graphe avec les arcs en tirets représente une ARPM de G_1 avec comme racine 0.

Nous nous interessons ici aux problèmes d'arborescence contraints (PAC) dont l'objectif est de trouver une ARPM sous des contraintes additionnelles. Nous présentons une contrainte globale dénommée `MinArborescence` pour modéliser ce type de problème. Notons que les coûts réduits de la programmation linéaire (PL) donne une borne inférieure de l'évolution du coût optimal si on force un arc qui n'était pas dans la solution optimale à y être. Le filtrage de la contrainte `MinArborescence` peut donc se faire avec ces derniers.

*Papier doctorant : V. R. Houndji[1,2] est auteur principal.

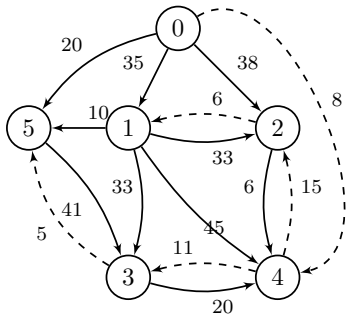


FIGURE 1 – Graphe G_1

Nous proposons une procédure pour améliorer la qualité des coûts réduits dans certains cas.

2 La contrainte globale `MinArborescence`

Nous modélisons l'arborescence avec une variable X_i pour chaque noeud i du graphe G représentant son prédécesseur. Le domaine initial de la variable X_i est donc l'ensemble des voisins (prédécesseurs) de i dans $G : j \in D(X_i) \equiv (j,i) \in E$. La contrainte `MinArborescence(X, w, r, K)` impose que l'ensemble des arcs $\{(X_i, i) : i \neq r\}$ est une arborescence valide enracinée en r avec $\sum_{i \neq r} w(X_i, i) \leq K$.

La cohérence de la contrainte `MinArborescence(X, w, r, K)` peut être vérifiée en calculant une ARPM $A(G)^\star$ enracinée en r et en vérifiant que $w(A(G)^\star) \leq K$. La valeur $w(A(G)^\star)$ est la borne inférieure exacte de la variable $K : K \geq w(A(G)^\star)$. Le filtrage des arcs peut être réalisé en se basant sur les coûts réduits de la PL $rc(i,j), \forall (i,j) \in E$. Pour un arc $(i,j) \notin A(G)^\star$, si $w(A(G)^\star) + rc(i,j) > K$, alors $X_i \leftarrow j$ est incohérent.

Dans la section suivante, nous donnons une propriété pour améliorer les coûts réduits de la PL $rc(i,j)$.

3 Amélioration des coûts réduits de la programmation linéaire

Utilisons les notations suivantes :

— $parent[S]$ est le plus petit cycle (voir [4]) qui contient le sous-ensemble $S \subseteq V \setminus \{r\}$;

— $bestTwoDiff[k]$ est la différence entre les deux plus petits poids des arcs qui entrent dans le noeud k ;

— u_S est la variable duale associée au sous-ensemble $S \subseteq V \setminus \{r\}$.

Supposons qu'il existe un chemin $\mathcal{P} = (j, \ldots, i)$ du noeud j vers le noeud i dans l'ARPM $A(G)^\star$ tel que $\forall k \in \mathcal{P} : parent[parent[k]] = \emptyset$. Alors $w(A(G)^\star_{i \to j}) \geq w(A(G)^\star) + rc(i,j) + \min_{k \in \mathcal{P} \setminus \{j\}} \{bestTwoDiff[k] - u^\star_{parent[k]}\}$. Nous nous référons à l'article [4] pour les preuves, les exemples et l'algorithme associé.

4 Résultats

Toutes les expériences ont été réalisées avec le solver open source OscaR [6]. Le code source ainsi que les instances utilisées sont disponibles en ligne [5]. Comme on peut s'y attendre, la décomposition basique avec le nombre exponentiel de contraintes n'est pas du tout compétitif. Pour avoir un meilleur modèle de base pour les expériences, nous avons proposé une contrainte légère nommée `Arborescence` (voir l'article [4]). Les tests ont été effectués sur une variante du PAC, l'ARPM-R [3]. Sur 100 instances avec $n = 50$, la taille de l'arbre de recherche est divisé, en moyenne, par ≈ 460 avec le filtrage basé sur les coûts réduits (par rapport au modèle de base). D'un autre côté, le filtrage basé sur les coûts réduits améliorés est intéressant sur certaines instances, en particulier les instances avec peu de cycles imbriqués.

5 Conclusion

Nous avons défini la contrainte `MinArborescence` avec un filtrage basé sur les coûts réduits pour les problèmes d'arborescence recouvrante de poids minimum. Nous avons également proposé une procédure pour améliorer les coûts réduits de la programmation linéaire dans certains cas.

Références

[1] J. Edmonds. Optimum Branchings. *Management Science*, 71B(4) :125–130, 1967.

[2] M. Fischetti and P. Toth. An efficient algorithm for min-sum arborescence problem on complete digraphs. *Network*, 29 :1520–1536, 1993.

[3] M. Fischetti and D. Vigo. A branch-and-cut algorithm for the resource-constrained minimum-weight arborescence problem. *Management Science*, 9 :55–67, 1997.

[4] V. R. Houndji and P. Schaus and M. N. Hounkonnou and L. Wolsey. The Weighted Arborescence Constraint In *Proceedings CPAIOR'2017*, 2017.

[5] V. R. Houndji and P. Schaus. CP4CAP : Constraint Programming for Constrained Arborescence Problem. Available from `https://bitbucket.org/ratheilesse/cp4cap`

[6] Oscar Team. "OscaR : Scala in OR". Available from `https://bitbucket.org/oscarlib/oscar`

MCSP3: la modélisation pour tous

Christophe Lecoutre

CRIL-CNRS UMR 8188,
Université d'Artois, F-62307 Lens, France
lecoutre@cril.fr

Résumé

Nous proposons MCSP3, une API de modélisation pour les problèmes combinatoires sous contraintes, actuellement "limitée" aux problèmes de satisfaction de contraintes (CSP) et problèmes d'optimisation sous contraintes (COP). MCSP3 est accompagné d'un compilateur qui permet de générer des instances définies au format intermédiaire XCSP3. La prise en main de cette API est rendue simple et naturelle, car elle est construite sur Java, sans doute aujourd'hui le langage le plus populaire en informatique. Grâce aux nouveaux concepts de la version 8 de Java, et notamment les lambda fonctions, l'écriture des modèles est compacte, particulièrement lisible, et pour l'aspect formel relativement proche des meilleurs langages dédiés. L'API propose de nombreuses méthodes, évitant dans une large mesure toute expression idiomatique (technique) propre au langage telle que l'utilisation de `new` pour la création d'objets, de tableaux, etc. Sur la base d'une expérience correspondant au développement de près d'une centaine de modèles à ce jour, nous pensons sincèrement que cette API est accessible au plus grand nombre. MCSP3 est une API de modélisation pour tous, disponible sur github.

1 Introduction

Le monde de la programmation par contraintes (PPC) pâtit du manque de standards, que ce soit pour la représentation des modèles ou pour celle des instances de problèmes combinatoires sous contraintes. Un modèle est une représentation paramétrée de la structure d'un ensemble d'instances pour un problème donné. Par exemple, il est possible de développer un modèle pour le problème des n-reines, où n représente une valeur entière jouant le rôle de paramètre, et de s'intéresser à la résolution d'une instance précise de ce problème, comme par exemple l'instance des 8-reines. Depuis trois ans, au sein du CRIL [1], nous développons une chaîne d'outils qui se revendique in fine être une approche globale, naturelle et cohérente permettant la modélisation et la résolution de problèmes sous contraintes. Dans cet article, nous présentons une brique importante de cette chaîne : MCSP3, une API de modélisation à la portée de tous.

Intéressons nous tout d'abord aux langages et outils de modélisation introduits dans la littérature au fil du temps. Parmi les langages marquants, certains sont dédiés à la programmation mathématique, comme AMPL (`http://www.ampl.com`) et GAMS (`http://www.gams.com`), et d'autres [2] à la programmation par contraintes, comme OPL [15], EaCL [14], NCL [16], ESRA [6], Zinc [4] et ESSENCE [7]. À ce jour, la situation reste malheureusement assez confuse pour l'utilisateur qui souhaite adopter une solution de modélisation PPC satisfaisante et pérenne.

Tout avait pourtant bien commencé. En effet, au début des années 70 [3], un langage étonnant voir le jour : Prolog. L'originalité de ce langage, basé sur le calcul des prédicats du premier ordre, son élégance et son aspect déclaratif engendre une grande effervescence, et ceci pendant de nombreuses années. La version 3 du langage [2] adopte un cadre de programmation logique par contraintes, et obtient un large succès, y compris auprès du grand public [3]. Au fil du temps, l'aspect "contraintes" se substitue à l'aspect "logique", et de nouveaux produits sont développés, les plus emblématiques étant certainement CHIP [5] et ILOG Solver. En terme de modélisation, OPL (Optimization Programming Language) [15] apparaît sans doute à ce

1. Avec le concours appréciable de collègues rattachés à des laboratoires de France et de Navarre (et aussi de Belgique ;-)).
2. Le langage Z (`http://vl.users.org`) permet également le développement de modèles plutôt élégants [11].
3. Il était fréquent que Prolog soit enseigné dans les formations universitaires en informatique.

moment-là comme étant le langage plus abouti, mais son caractère propriétaire le prive d'une appropriation par l'ensemble de la communauté.

Après les belles aventures Prolog et OPL, la communauté se tourne progressivement vers le développement de solveurs ouverts, le plus souvent intégrant une interface de modélisation dédiée, tels que par exemple Gecode [13] et Choco [10]. Cependant, le besoin d'uniformisation se faisant sentir de plus en plus fortement, de nouveaux langages de modélisation sont proposés, notamment au cours de la dernière décennie avec Essence et MiniZinc. Si Essence est assurément de belle facture, la suite qui l'accompagne reste, nous semble-t-il, assez difficile d'accès. En ce qui concerne Minizinc, nous n'avons jamais été convaincus par les choix effectués par les développeurs de cette suite (nous développerons nos arguments dans un article à venir).

Pour la représentation d'instances de problèmes, il est possible de se tourner vers des formats à plat tels que XCSP 2.1 [12] ou FlatZinc, ou le format intermédiaire XCSP3 [1] que nous avons introduit récemment. XCSP3 a l'avantage d'être très complet, lisible et structuré. La figure 1 offre une vision synthétique de la situation actuelle concernant langages et formats de modélisation.

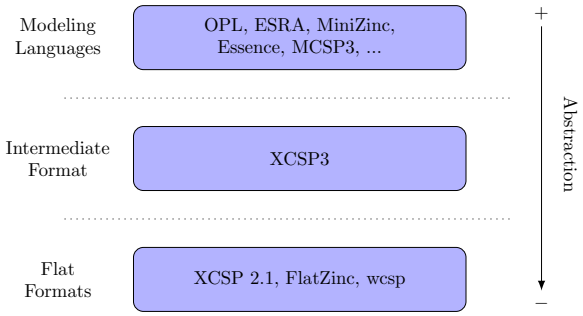


FIGURE 1 – Langages de modélisation et formats

Ce constat posé, nous avons cherché à développer de façon posée et cohérente une chaîne de production complète et intégrée pour le domaine de la PPC. Les deux principaux ingrédients sont :

— MCSP3 : une interface (API) basée sur Java permettant de modéliser les problèmes sous contraintes de façon déclarative et naturelle ;
— XCSP3 : un format intermédiaire utilisé pour représenter les instances de problèmes, tout en préservant leurs structures.

Comme cela est visible sur la figure 2, confronté à un problème à résoudre, l'utilisateur doit procéder comme suit :

1. écrire un modèle en utilisant l'interface MCSP3 construite sur Java 8 ;
2. fournir les données, au format JSON, correspondant aux instances précises à résoudre ;
3. compiler modèle et données afin de générer des fichiers au format intégré XCSP3 ;
4. résoudre les instances XCSP3 en utilisant des solveurs tels que Choco ou OscaR [9].

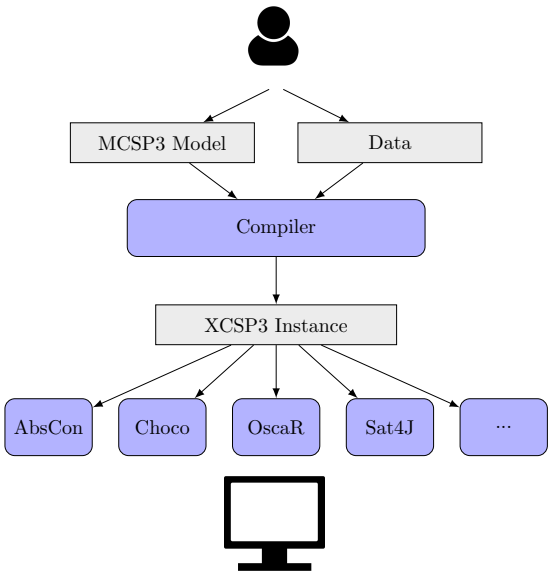


FIGURE 2 – Chaîne de production complète MCSP3-XCSP3.

La chaîne de production complète, MCSP3-XCSP3, a de nombreux avantages :

— Java, JSON et XML sont des technologies éprouvées de premier plan ;
— utiliser Java 8 pour la modélisation [4] évite à l'utilisateur d'apprendre encore (et toujours) un nouveau langage ;
— utiliser JSON pour les données offre une notation simple, unifiée et facile à lire aussi bien pour l'humain que pour la machine :
— utiliser XML pour les instances (mais avec une granularité raisonnable) permet une représentation simple et lisible, de nouveau que ce soit pour l'humain ou la machine.

Il peut sembler étonnant que JSON et XML soient tous les deux sollicités le long de la chaîne. En fait, nous sommes convaincus que JSON est le format le plus approprié pour les données tandis que XML est

4. L'introduction des lambda fonctions en Java 8 a été l'élément déclencheur qui nous a permis de développer une interface sobre et naturelle.

mieux adapté pour la représentation des instances (utiliser JSON est envisageable, mais possède quelques inconvénients comme cela est indiqué dans l'une des appendices des spécifications XCSP3 [1]).

La version courante de MCSP3 est une "release candidate", mais elle est néanmoins déjà robuste car testée sur une centaine de modèles. L'API (avec son compilateur) est disponible sur github : https://github.com/xcsp3team/XCSP3-Java-Tools, dans le package `modeler`. Une documentation plus fournie se trouve dans [8].

2 Illustration

Nous proposons d'illustrer l'interface MCSP3 avec un cas d'étude, le problème d'allocation d'entrepôt défini sur CSPLib–Problem 034 comme suit :

"In the Warehouse Location problem (WLP), a company considers opening warehouses at some candidate locations in order to supply its existing stores. Each possible warehouse has the same maintenance cost, and a capacity designating the maximum number of stores that it can supply. Each store must be supplied by exactly one open warehouse. The supply cost to a store depends on the warehouse. The objective is to determine which warehouses to open, and which of these warehouses should supply the various stores, such that the sum of the maintenance and supply costs is minimized."

FIGURE 3 – Un entrepôt.

Du point de vue de l'utilisateur, modéliser un problème nécessite trois étapes :

1. Définir la structure (type) des paramètres du problème. Une instance du problème est-elle caractérisée par un simple entier, une séquence d'entiers, un ou plusieurs tableaux de valeurs numériques, voire symboliques ? etc.
2. Définir la structure du modèle en utilisant un langage approprié. Ici, il s'agit de MCSP3.
3. Définir un jeu d'instances en produisant un jeu de données respectant la structure préalablement définie.

Après réflexion, nous découvrons que notre problème nécessite trois paramètres. Le premier `fixedCost` représente le coût fixe entier associé à l'ouverture d'un entrepôt. Le second, `warehouseCapacities` représente le nombre de magasins que chaque entrepôt peut alimenter. Le troisième, `storeSupplyCosts` représente le coût pour alimenter chaque magasin avec chaque entrepôt. La structure des paramètres est donc donnée ici par un entier, un tableau d'entiers et un tableau à deux dimensions d'entiers. Un exemple de données pour une instance spécifique est donné par le fichier suivant "warehouse.json" contenant :

```
{
  "fixedCost": 30,
  "warehouseCapacities": [1,4,2,1,3],
  "storeSupplyCosts": [
    [100,24,11,25,30],[28,27,82,83,74],
    [74,97,71,96,70],[2,55,73,69,61],
    [46,96,59,83,4],[42,22,29,67,59],
    [1,5,73,59,56],[10,73,13,43,96],
    [93,35,63,85,46],[47,65,55,71,95]
  ]
}
```

Un modèle possible en MCSP3 est :

```
class Warehouse implements ProblemAPI {
  int fixedCost;
  int[] warehouseCapacities;
  int[][] storeSupplyCosts;

  public void model() {
    int nWarehouses = warehouseCapacities.length;
    int nStores = storeSupplyCosts.length;

    Var[] s = array("s", size(nStores),
      dom(range(nWarehouses)),
      "s[i] is the warehouse supplier of store i");

    Var[] c = array("c", size(nStores),
      i -> dom(storeSupplyCosts[i]),
      "c[i] is the cost of supplying store i");

    Var[] o = array("o", size(nWarehouses),
      dom(0, 1), "o[i] is 1 if the warehouse i is open");

    forall(range(nWarehouses),
      i -> atMost(s, i, warehouseCapacities[i]));
    forall(range(nStores),
      i -> element(o, s[i], 1));
    forall(range(nStores),
      i -> element(storeSupplyCosts[i], s[i], c[i]));

    int[] coeffs = vals(repeat(1,nStores),
       repeat(fixedCost,nWarehouses));
    minimize(SUM, vars(c,o), coeffs)
      .note("minimizing the overall cost");
  }
}
```

Comme vous pouvez l'observer, définir un modèle nécessite donc d'écrire une classe implémentant l'interface `ProblemAPI`. Cette classe doit intégrer des champs correspondant aux données du fichier JSON (le chargement se fait automatiquement au moment de la compilation). Il est ensuite nécessaire d'implanter la méthode `model()`, en intégrant la déclaration des variables, contraintes et objectifs du problème. Pour notre problème, nous déclarons trois tableaux de variables : la taille des tableaux est définie par appel à la méthode `size()`, et le domaine de chaque variable est défini par appel à la méthode `dom()`. Il est intéressant de constater qu'il est possible de définir un domaine spécifique pour chaque variable d'un tableau en utilisant une lambda fonction. Pour poster des groupes de contraintes, on utilise la méthode `forall`. Le premier groupe signifie que pour tout entrepôt d'indice i, on souhaite qu'au plus `warehouseCapacities[i]` variables de s prennent la valeur i. Les deux autres groupes permettent de poster des contrainte `element`. Pour finir, l'objectif à minimiser est une somme de variables coefficientées. La méthode `vars()` permet de définir un tableau de variables en collectant celles qui sont passées en paramètres (y compris lorsqu'elles sont déjà définies dans des tableaux).

De façon générale, il existe dans l'API de nombreuses surcharges de méthodes pour définir les tableaux de variables (jusqu'à une dimension 5), les domaines, les groupes (jusque 5 boucles imbriquées en utilisant la combinaison de méthodes `range()`). Actuellement, dans la version courante (release candidate), la vingtaine de contraintes appartenant à XCSP3-core est utilisable. Cela offre déjà un grand pouvoir d'expression, et nous a d'ailleurs permis de modéliser une centaine de problèmes.

3 Conclusion

MCSP3 est une API de modélisation qui permet de représenter les problèmes combinatoires sous contraintes. Basée sur Java 8, elle permet à l'utilisateur de construire aisément les modèles et de les compiler en instances XCSP3. Les versions futures intégreront d'autres cadres (WCSP, par exemple), d'autres contraintes, les variables réelles et ensemblistes, la réification, les annotations, etc. Comparé à la "release candidate" courante de MCSP3, qui a nécessité un effort de développement particulièrement important, ces futures extensions ne présentent pas de difficultés techniques majeures.

Références

[1] F. Boussemart, C. Lecoutre, and C. Piette. XCSP3 : An integrated format for benchmarking combinatorial constrained problems. Technical Report arXiv :1611.03398, CRIL, 2016.

[2] A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7) :69–90, 1990.

[3] A. Colmerauer, H. Kanoui, P. Roussel, and R. Pasero. Un système de communication homme-machine. Technical report, Groupe de recherche en Intelligence Artificielle, Marseille, 1973.

[4] M. Garcia de la Banda, K. Marriott, R. Rafeh, and M. Wallace. The modelling language Zinc. In *Proceedings of CP'06*, pages 700–705, 2006.

[5] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of FGCS'88*, pages 693–702, 1988.

[6] P. Flener, J. Pearson, and M. Ågren. Introducing ESRA, a relational language for modelling combinatorial problems. In *LOPSTR'03 : Revised Selected Papers*, pages 214–232, 2004.

[7] A. Frisch, M. Grum, C. Jefferson, B. Martinez Hernandez, and I. Miguel. The design of ESSENCE : A constraint language for specifying combinatorial problems. In *Proceedings of IJCAI'07*, pages 80–87, 2007.

[8] C. Lecoutre. MCSP3 : Easy modeling for everybody. Technical Report Release Candidate, available from https://github.com/xcsp3team/XCSP3-Java-Tools, CRIL, 2017.

[9] OscaR Team. OscaR : Scala in OR, 2012. Available from https://bitbucket.org/oscarlib/oscar.

[10] C. Prud'homme, J.-G. Fages, and X. Lorca. Choco, 2016. Available from http://www.choco-solver.org.

[11] G. Renker and H. Ahriz. Building models through formal specification. In *Proceedings of CPAIOR'04*, pages 395–401, 2004.

[12] O. Roussel and C. Lecoutre. XML representation of constraint networks : Format XCSP 2.1. Technical Report arXiv :0902.2362, CoRR, 2009.

[13] C. Schulte, G. Tack, and M.Z. Lagerkvist. Gecode : A generic constraint development environment. http ://www.gecode.org, 2006.

[14] E. Tsang, P. Mills, R. Williams, J. Ford, and J. Borrett. A computer-aided constraint programming system. In *Proceedings of PACLP'99*, pages 81–93, 1999.

[15] P. van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.

[16] J. Zhou. Introduction to the constraint language NCL. *Journal of Logic Programming*, 45(1-3) :71–103, 2000.

Combinaison de *nogoods* extraits au redémarrage

Gaël Glorian * **Frédéric Boussemart** **Jean-Marie Lagniez**
Christophe Lecoutre **Bertrand Mazure**

CRIL-CNRS UMR 8188, Université d'Artois, F-62307 Lens, France
{glorian, boussemart, lagniez, lecoutre, mazure}@cril.fr

Résumé

Dans cet article, nous nous intéressons à l'enregistrement de *nogoods*, instanciations partielles globalement incohérentes, pouvant être extraits systématiquement lors du redémarrage d'un algorithme complet (avec retour-arrière) de résolution CSP (problème de satisfaction de contraintes). Plus précisément, dans ce contexte, nous proposons plusieurs techniques de simplification et de combinaison de *nogoods*, dans le but d'accroître leur capacité de filtrage. La base de notre approche est une généralisation des *nld-nogoods* correspondant au concept introduit récemment d'*increasing-nogoods*. Nous proposons plusieurs algorithmes portant sur des combinaisons de sous-ensembles de *nogoods* identifiés de manière dynamique. Les similarités entre les différents *increasing-nogoods* permettent un meilleur élagage de l'arbre de recherche, notamment grâce à l'exploitation d'équivalences entre décisions. Nous proposons aussi quelques pistes d'amélioration, notamment un système de sentinelles et la génération de *nld-nogoods* à la volée. Nos résultats préliminaires montrent l'intérêt de notre approche pour certains problèmes.

Abstract

In this paper, we exploit nogoods, partial unsatisfiable instantiations, extracted from a restart-based search engine at the end of each run when the current cutoff value is reached. More precisely, in that context, we propose several simplification and combination techniques related to nogoods, in order to increase the filtering efficiency. The root of our approach is a generalization of nld-nogoods corresponding to the concept newly introduce of increasing-nogoods. We propose various algorithms relating to dynamically identified nogoods subsets combinations. Therefore, the search tree benefits from a better pruning thanks to similarities existing between *increasing-nogoods*, especially the equivalence between decisions. We also suggest some improvements tracks, in particular a sentinel system and on the fly nld-nogoods production. Our preliminary results show that our approach works well for several problems.

*Papier doctorant : Gaël Glorian est auteur principal.

1 Introduction

L'apprentissage de *nogoods* est un thème qui a été introduit dans les années 90 [2, 15, 4] pour la résolution de problèmes de satisfaction de contraintes. Les *nogoods* classiques, dits standards, sont des instanciations partielles ne pouvant mener à aucune solution. Ils ont été assez rapidement utilisés pour gérer l'explication [5, 7] de valeurs supprimées au cours de la recherche et de la propagation de contraintes. Ils ont ensuite été généralisés [8] en permettant la combinaison d'assignations de variables (décisions positives) et de réfutations de valeurs (décisions négatives). L'intérêt pratique des *nogoods* (généralisés) a été revisité par les travaux portant sur la génération paresseuse de clauses [3].

Les *nogoods* peuvent également être utiles dans un contexte de redémarrage régulier d'un algorithme de recherche arborescent. Il est en effet possible d'identifier [10] sur la dernière branche (celle qui est la plus à droite) un ensemble de *nogoods* représentant la partie de l'espace de recherche qui vient d'être explorée. Par le fait de simplement enregistrer ces *nogoods*, appelés *nld-nogoods* (réduits), on a la garantie de ne jamais explorer de nouveau les mêmes sous-arbres. Certaines extensions de ces travaux ont porté sur l'élimination de symétries [11, 13] et l'exploitation du caractère croissant des *nld-nogoods* [12], appelés de ce fait *increasing-nogoods*.

Dans cet article, nous proposons plusieurs techniques de simplification et de combinaison d'*increasing-nogoods*, nous permettant d'accroître leur capacité de filtrage. En analysant dynamiquement certaines combinaisons de sous-ensembles de *nogoods*, et en exploitant des formes d'équivalence entre décisions, nous montrons que l'arbre de recherche peut être mieux élagué. Nous identifions également quelques pistes prometteuses permettant de renforcer l'efficacité du processus de détection.

Cet article est structuré comme suit. Dans un premier temps, nous présentons les *nld-nogoods* ainsi que les *increasing-nogoods*. Puis, nous introduisons quelques techniques de combinaisons (avec leurs algorithmes) : combinaisons de décisions négatives, combinaisons par équivalence d'alpha, et combinaisons par équivalence de décisions négatives. Après avoir présenté quelques résultats expérimentaux, nous concluons et évoquons quelques pistes prometteuses.

2 Préliminaires

Un réseau fini de contraintes P est un couple $(\mathcal{X}, \mathcal{C})$, où $\mathcal{X}$ est un ensemble de variables et $\mathcal{C}$ un ensemble de contraintes. À chaque variable $x \in \mathcal{X}$ est associé un domaine, noté $dom(x)$ qui contient un ensemble fini de valeurs. Chaque contrainte $c \in \mathcal{C}$ porte sur un nombre fini de variables appelé la portée de c, noté $scp(c)$ et est définie formellement par une relation notée $rel(c)$ qui contient les tuples autorisés pour les variables de la portée, c'est-à-dire les combinaisons qui satisfont c. L'arité d'une contrainte c correspond au nombre de variables sur lesquelles la contrainte porte ($|scp(c)|$). Une solution de P est une instanciation de toutes les variables de P satisfaisant toutes les contraintes de P. Le problème CSP est un problème de décision qui consiste à déterminer si un réseau de contraintes donné admet une solution.

Un *nogood* est une instanciation partielle qui ne peut être prolongée vers une solution. L'intérêt des *nogoods* est d'éviter le phénomène de *thrashing*, c'est-à-dire d'explorer de manière répétée les mêmes sous-arbres incohérents. Deux approches classiques existent pour les identifier et les enregistrer : au cours de la recherche, ou au redémarrage. Dans cet article, nous allons nous placer dans le contexte d'un algorithme complet de résolution avec retour-arrière à branchement binaire et enregistrement de *nogoods* au redémarrage.

2.1 Nogood recording from restart

Les *nld-nogoods* [9] (*negative last decision nogoods*) sont extraits au redémarrage à partir de la dernière branche Σ de l'arbre de recherche. $\Sigma = \langle \delta_1, ..., \delta_m \rangle$ avec δ_i une décision positive, c'est-à-dire une assignation ($x = a$), ou négative, une réfutation ($x \neq a$). Ils représentent tous les conflits obtenus au cours du *run* précédent (i.e., depuis le dernier redémarrage). Une *nld-sous-séquence* est une séquence de décisions $\langle \delta_1, ..., \delta_j \rangle$ où δ_j est une décision négative. Pour toute nld-sous-séquence $\Sigma' = \langle \delta_1, ..., \delta_j \rangle$ de Σ, l'ensemble $\{\delta_1, ..., \neg\delta_j\}$ est un *nld-nogood* et l'ensemble $pos(\Sigma') \cup \{\neg\delta_j\}$, où $pos(\Sigma')$ correspond aux décisions positives de Σ', est un *nld-nogood* réduit. Il est connu que tous les *nld-nogoods* réduits extraits de la dernière branche de l'arbre de recherche subsument tous les *nld-nogoods* réduits qui pourraient être extraits des branches précédemment explorées.

Exemple. Considérons que l'algorithme soit sur le point d'effectuer un redémarrage, et que nous ayons l'arbre de recherche illustré par la figure 1.

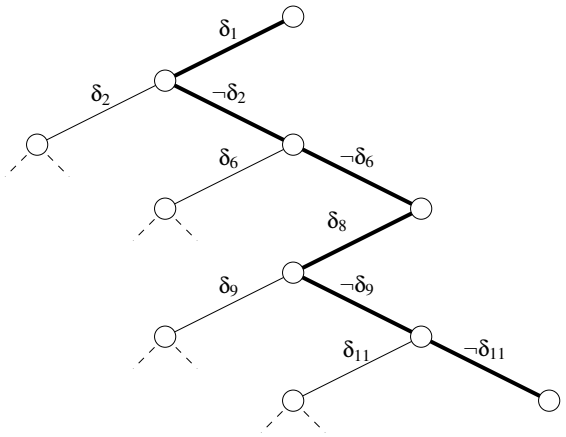


FIGURE 1 – Arbre de recherche avant redémarrage

La dernière branche de l'arbre de recherche est $\Sigma = \{\delta_1, \neg\delta_2, \neg\delta_6, \delta_8, \neg\delta_9, \neg\delta_{11}\}$. Nous pouvons en extraire les *nld-nogoods* suivants (à gauche ci-dessous) ainsi que les *nld-nogoods* réduits associés (à droite ci-dessous) :

$$\Delta_1 = \{\delta_1, \delta_2\} \qquad \Delta_{1r} = \{\delta_1, \delta_2\}$$
$$\Delta_2 = \{\delta_1, \neg\delta_2, \delta_6\} \qquad \Delta_{2r} = \{\delta_1, \delta_6\}$$
$$\Delta_3 = \{\delta_1, \neg\delta_2, \neg\delta_6, \delta_8, \delta_9\} \qquad \Delta_{3r} = \{\delta_1, \delta_8, \delta_9\}$$
$$\Delta_4 = \{\delta_1, \neg\delta_2, \neg\delta_6, \delta_8, \neg\delta_9, \delta_{11}\} \quad \Delta_{4r} = \{\delta_1, \delta_8, \delta_{11}\}$$

Nous pouvons remarquer qu'il existe des similitudes entre les différents *nld-nogoods* extraits lors d'un même redémarrage, ils sont dits croissants. Les *increasing-nogoods* [12, 13] présentés dans la section suivante exploitent ce caractère croissant et permettent de représenter les *nld-nogoods* réduits extraits à la fin d'un *run* sous la forme d'une (seule) contrainte globale.

2.2 Increasing nogoods

Les *increasing-nogoods* (*IncNG*) [12, 13] sont une extension des *nld-nogoods* réduits. À chaque redémarrage une seule contrainte globale *IncNG* est ajoutée au réseau. Celle-ci représente tous les *nld-nogoods* réduits qui auraient pu être extraits lors de la recherche. L'idée est de mettre à disposition une structure compacte ainsi qu'un filtrage supérieur aux *nld-nogoods* réduits traités

indépendamment.

Pour passer d'un ensemble de *nld-nogoods* réduits à une contrainte *IncNG*, il faut commencer par les transformer en *nld-nogoods* réduits *dirigés*. Soit un *nld-nogood* réduit $\{\delta_i, \delta_j\}$, le *nld-nogood* réduit dirigé correspondant s'écrit $\delta_i \Rightarrow \neg\delta_j$.

Exemple. Revisitons l'arbre de recherche de l'exemple précédent en précisant les décisions (assignations et réfutations). Sur la branche $\Sigma = \langle\, x_2 = 1,\, x_3 \neq 0,\, x_4 \neq 1,\, x_5 = 2,\, x_1 \neq 1,\, x_6 \neq 2 \,\rangle$ avec $dom(x_i) = \{0,1,2\}$, nous pouvons extraire l'ensemble suivant de *nld-nogoods* réduits :

$$\begin{aligned} ng_0 &\equiv \neg(\ x_2 = 1 \wedge x_3 = 0) \\ ng_1 &\equiv \neg(\ x_2 = 1 \wedge x_4 = 1) \\ ng_2 &\equiv \neg(\ x_2 = 1 \wedge x_5 = 2 \wedge x_1 = 1) \\ ng_3 &\equiv \neg(\ x_2 = 1 \wedge x_5 = 2 \wedge x_6 = 2) \end{aligned}$$

qui sous forme dirigée s'écrivent :

$$\begin{aligned} ng_0 &\equiv x_2 = 1 \Rightarrow x_3 \neq 0 \\ ng_1 &\equiv x_2 = 1 \Rightarrow x_4 \neq 1 \\ ng_2 &\equiv x_2 = 1 \wedge x_5 = 2 \Rightarrow x_1 \neq 1 \\ ng_3 &\equiv x_2 = 1 \wedge x_5 = 2 \Rightarrow x_6 \neq 2 \end{aligned}$$

Dans [12], les auteurs ont montré que l'ensemble des *nld-nogoods* réduits dirigés extraits d'une branche sont croissants (*increasing*), c'est-à-dire que $LHS(ng_i) \subseteq LHS(ng_{i+1})$ où *LHS* (*left hand side*) désigne la partie gauche de l'implication (similairement *RHS* désigne la partie droite).

$$\begin{aligned} ng_0 &\equiv x_2 = 1 \Rightarrow x_3 \neq 0 \\ ng_1 &\equiv LHS(ng_0) \Rightarrow x_4 \neq 1 \\ ng_2 &\equiv LHS(ng_1) \wedge x_5 = 2 \Rightarrow x_1 \neq 1 \\ ng_3 &\equiv LHS(ng_2) \Rightarrow x_6 \neq 2 \end{aligned}$$

Les *nld-nogoods* réduits dirigés ainsi obtenus peuvent s'écrire sous forme de séquence de décisions en éliminant les parties gauches redondantes :

$$\Sigma = \langle\, x_2 = 1,\, x_3 \neq 0,\, x_4 \neq 1,\, x_5 = 2,\, x_1 \neq 1,\, x_6 \neq 2 \,\rangle$$

Nous remarquons que cela correspond à la séquence de décision initiale, celle extraite de l'arbre de recherche. Pour construire un *IncNG* il suffit donc de retenir la dernière branche avant le redémarrage.

Soit $\Lambda = \langle ng_0, \ldots, ng_t \rangle$ une séquence composée de *nld-nogoods* croissants. Si $LHS(ng_i)$ contient deux décisions positives pouvant encore être falsifiées alors les *nogoods* ng_j tel que $j \geq i$ sont nécessairement arc-cohérents (ou GAC pour Generalized Arc Consistency) car les parties gauches des *nogoods* plus grands subsument celles des plus petits.

Pour filtrer la contrainte *IncNG*, deux indices α et β sont utilisés. Ils correspondent aux deux décisions positives non assignées les plus à gauche dans la séquence (pouvant encore être falsifiées). Celles-ci sont surveillées ainsi que toutes les décisions négatives se situant entre α et β.

$$\Sigma = \langle\, \overbrace{\underbrace{\delta_1}_{\alpha}, \neg\delta_2, \underbrace{\delta_3}_{\beta}}^{\text{Watched}}, \delta_4, \neg\delta_5, \neg\delta_6 \rangle$$

Il existe trois situations principales d'appel à l'algorithme de filtrage (algorithme 1) :

1. une décision négative contenue entre α et β est falsifiée, cela force δ_α à être faux, par conséquent tous les *nogoods* contenus dans la contrainte sont falsifiés ;
2. la décision positive désignée par α est satisfaite : nous forçons toutes les parties droites qui ne contiennent que δ_α dans leur partie gauche à être vraies c'est-à-dire toutes les décisions négatives contenues entre α et β et nous recherchons la prochaine décision positive non assignée ;
3. la décision positive désignée par β est satisfaite : ceci est semblable au cas précédent, nous recherchons la prochaine décision positive non assignée.

Algorithme 1 : FilterIncNG(Σ)

Data : $m = |\Sigma|$ where Σ is an *IncNG*
1 UpdateAlpha();
2 **if** $m \neq 0 \wedge \beta \neq m$ **then** UpdateBeta();
3 **if** $m = 0$ **then** delete constraint;

Les algorithmes 2 et 3 qui ont été proposés par [12] ont un fonctionnement relativement similaire. Si δ_α est satisfait, un nouvel α est trouvé, l'algorithme est appelé de nouveau pour voir si δ_α est satisfaite jusqu'à arriver à un point fixe ou que la contrainte soit entièrement traitée (de la même manière pour β). La fonction *watchFollowDec* permet de trouver la prochaine décision positive à partir de β si elle existe et de surveiller toutes les décisions négatives rencontrées. Si une décision négative falsifiée est rencontrée, m, qui représente la taille de l'*IncNG* traité, est mis à zéro, ce qui correspond à la désactivation de cet *IncNG*.

3 Combinaison d'*increasing-nogoods*

Les techniques existantes de traitement de *nogoods*, que ce soient les *nld-nogoods* ou les *increasing-nogoods*, se

Algorithme 2 : UpdateAlpha()

```
1  if δα is satisfied then
2      unsubscribe δα;
3      for i ∈ [α+1, β) do
4          if Neg(δi) then // true if δi is negative
5              satisfy δi;
6              unsubscribe δi;
7          end
8      end
9      if β = m then m ← 0 ; return ;
10     α ← β;
11     watchFollowDec();
12     if m ≠ 0 then UpdateAlpha();
13 else
14     for i ∈ [α+1, β) do
15         if Neg(δi) ∧ δi is falsified then
16             falsify δα;
17             m ← 0;
18             return ;
19         end
20     end
21 end
```

Algorithme 3 : UpdateBeta()

```
1  if δβ is satisfied then
2      unsubscribe δβ;
3      watchFollowDec();
4      if m ≠ 0 then UpdateBeta();
5  end
```

limitent à elles-mêmes. Ces méthodes ne tirent pas partie des autres informations disponibles, à savoir l'état des variables (c'est-à-dire, leur domaine) ainsi que les autres *nogoods*. Nous proposons donc dans cette section trois méthodes utilisant à bon escient ces informations. La première permet de continuer à traiter les *increasing-nogoods* indépendamment mais les fait toutefois interagir sur la base des domaines des variables qui les composent. Les deux autres regroupent les *increasing-nogoods* par sous-ensembles, soit en fonction de leurs décisions positives (δ_α), soit en fonction de leurs décisions négatives (celles qui sont comprises entre alpha et beta).

3.1 Combinaison de décisions négatives

Le but de la combinaison de décisions négatives est de sécuriser le fait qu'une assignation ou une suppression ne cause pas de conflit direct avec la base de *nogoods*, c'est-à-dire avant que l'*IncNG* relatif à la combinaison entre en conflit. Cela permet de déceler les conflits en amont et par conséquent d'augmenter le pouvoir de filtrage des *increasing-nogoods*.

Exemple. Soit le *IncNG* suivant :

$$ng_i = \langle\ x_2 = 1, x_3 \neq 2, x_3 \neq 4,\ x_5 = 3 \rangle$$

Supposons qu'il soit dans son état initial, c'est-à-dire $\alpha \leftarrow x_2 = 1$ et $\beta \leftarrow x_5 = 3$ et que les variables aient toutes le même domaine $dom(x_i) = \{1,2,3,4\}$. Si x_2 est assigné à 1, alors il est possible de supprimer les valeurs 2 et 4 du domaine de x_3 (voir la situation 2 de l'algorithme 1). Si le domaine de x_3 ne contenait plus que ces deux valeurs alors il y a conflit. Ce conflit aurait pu être détecté avant, voire évité, si la valeur 1 avait été supprimée du domaine de x_2 lorsque le domaine de x_3 a été réduit à $\{2,4\}$.

Par souci de simplicité, nous considérons que pour *ng*, un *IncNG* donné, les valeurs de alpha et beta sont accessibles par $\alpha(ng)$ et $\beta(ng)$. Nous pouvons alors définir `diffValues`(ng,x), la fonction qui retourne les valeurs impliquées dans une décision négative de *ng* impliquant x et située entre $\alpha(ng)$ et $\beta(ng)$. De même, `diffVars`(ng) est la fonction qui retourne l'ensemble des variables impliquées dans une décision négative de *ng* située entre $\alpha(ng)$ et $\beta(ng)$. Par exemple, si nous considérons l'*IncNG* $ng_{ex} = \langle\ x_2 = 1, x_3 \neq 2, x_3 \neq 4,\ x_5 = 3 \rangle$ avec $\alpha(ng_{ex}) = x_2$ et $\beta(ng_{ex}) = x_5$, alors `diffVars`$(ng_{ex}) = \{x_3\}$ et `diffValues`$(ng_{ex}, x_3) = \{2,4\}$.

Algorithme 4 : checkNegativeDecisions(ng)

```
   Data : Let ng be an IncNG
1  foreach x ∈ diffVars(ng) do
2      if diffValues(ng, x) ⊇ dom(x) then
3          falsify α(ng);
4      end
5  end
```

L'algorithme 4 réalise ce filtrage qui consiste donc à effectuer une inférence (réfuter la valeur impliquée dans $\alpha(ng)$) chaque fois qu'un conflit est susceptible de se produire. Notre approche, bien que se limitant à l'analyse des *increasing-nogoods* de manière indépendante, offre une capacité de filtrage renforcée. Dans la section suivante, nous proposons une nouvelle variation de ce principe par analyse de la base complète d'*increasing-nogoods*. L'algorithme 4 a une complexité dans le pire cas en $O(sn)$ où s est la taille du plus grand *IncNG* (majoré par dn où d est la taille du plus grand domaine) et n le nombre de variables.

3.2 Combinaison par équivalence d'alpha

Dans cette partie, nous étendons le principe présenté précédemment à des ensembles d'*increasing-nogoods*. Pour cela, nous partitionnons l'ensemble des *increasing-nogoods* en fonction des valeurs des δ_α : les *increasing-nogoods* de même valeur se situent dans le même groupe. Il faut donc maintenir ces groupes de manière dynamique, c'est-à-dire les mettre à jour à chaque modification d'un alpha, lors d'un filtrage ou d'un retour-arrière. En raisonnant avec ces groupes, nous obtenons une capacité de filtrage renforcée (englobant le cas précédent).

Exemple. Soit $dom(x_i) = \{0,1,2,3\}$,

$$IncNG_0 \equiv \quad \dots\ ,\ x_6 \neq 2,\ \underbrace{x_2 = 1}_{\alpha},\ x_1 \neq 3,\ \underline{x_3 \neq 1},\ \dots$$

$$IncNG_1 \equiv \quad \dots\ ,x_2 \neq 0,\ x_1 \neq 2,\ \underbrace{x_2 = 1}_{\alpha},\ \underline{x_3 \neq 0},\ \dots$$

$$IncNG_2 \equiv \quad \dots\ ,\underbrace{x_2 = 1}_{\alpha},\ \underline{x_3 \neq 2},\ x_6 \neq 1,\ x_8 \neq 3,\ \dots$$

Sur l'exemple, nous pouvons constater que $x_2 = 1$ est le δ_α commun à un groupe de trois *increasing-nogoods*. En regardant les décisions négatives qui suivent ces trois occurrences de δ_α (la valeur de beta n'est pas importante pour notre illustration), nous remarquons que trois valeurs différentes pour x_3 apparaissent dans les décisions négatives surveillées (avant le beta qui n'est pas représenté). Cela signifie que si x_2 venait à être assigné à 1 la seule valeur restante dans le domaine de x_3 serait 3. De ce fait, si la valeur 3 a déjà été retirée du domaine de x_3, il faut alors absolument empêcher x_2 de pouvoir être assignée à 1. Pour empêcher cela, tous les *increasing-nogoods* du groupe vont être désactivés (car forcées à être toujours vérifiés après avoir supprimé la valeur 1 du domaine de x_2). L'algorithme 5 réalise ce filtrage.

Algorithme 5 : checkNegativeDecGroup(`ngGroup`)

```
Data : Let ngGroup be a set of increasing-nogoods
       with a common δα
1 X = ⋃ {x_i ∈ diffVars(ng_j) | ng_j ∈ ngGroup};
2 foreach x ∈ X do
3 |   V=⋃{v_i∈diffValues(ng_j,x)|ng_j∈ngGroup};
4 |   if V ⊇ dom(x) then
5 |   |   falsify α(ngGroup);
6 |   end
7 end
```

L'algorithme 5 crée en premier lieu l'ensemble `X` qui contient toutes les variables apparaissant entre alpha et beta du groupe passé en paramètre de l'algorithme. De plus, il constitue pour chaque variable contenue dans `X` un ensemble de valeurs `V`. L'ensemble de valeurs `V` ainsi constitué est comparé au domaine de la variable relative, si ceux-ci sont équivalents, la décision pointée par alpha du groupe *ngGroup* est falsifié en vue d'éviter un conflit possible dans la suite de la recherche. L'algorithme 5 a une complexité dans le pire cas en $O(nsg)$ où n le nombre de variables total, s est la taille du plus grand *IncNG* et g le nombre d'*increasing-nogoods* dans la base.

Nous allons voir qu'il est possible de traiter les *increasing-nogoods* ayant la même variable pointée par alpha mais avec des valeurs différentes et ainsi extraire des sous-groupes ayant des décisions négatives communes de ces ensembles.

3.3 Combinaison par équivalence de décisions négatives

De la même manière que précédemment, les *increasing-nogoods* sont ici aussi traités par ensemble. La différence réside dans le fait que nous regroupons ceux qui ont un alpha de variable commune et surveillent une même décision négative (que nous appelons pivot) située entre les alphas et betas courants. Si toutes les valeurs restantes dans le domaine de la variable commune des alphas sont présentes dans le groupe, alors la décision négative pivot doit être vérifiée.

Exemple. Soit $\forall i \in \mathbb{N},\ dom(x_i) = \{0,1,2,3\}$,

$$IncNG_0 \equiv \quad \dots\ ,\ x_6 \neq 2,\ \underbrace{x_2 = 1}_{\alpha},\ x_1 \neq 3,\ \underline{x_3 \neq 1},\ \dots$$

$$IncNG_1 \equiv \quad \dots\ ,x_7 \neq 0,\ x_1 \neq 2,\ \underbrace{x_2 = 0}_{\alpha},\ \underline{x_3 \neq 1},\ \dots$$

$$IncNG_2 \equiv \quad \dots\ ,\underbrace{x_2 = 2}_{\alpha},\ \underline{x_3 \neq 1},\ x_6 \neq 1,\ x_8 \neq 3,\ \dots$$

$$IncNG_3 \equiv \quad \dots\ ,\ x_4 \neq 3,\underbrace{x_2 = 3}_{\alpha},\ \underline{x_3 \neq 1},\ x_1 \neq 1,\ \dots$$

Sur l'exemple nous pouvons voir que $x_3 \neq 1$ est une décision négative surveillée commune à l'ensemble d'*increasing-nogoods*. En regardant les alphas nous remarquons que les quatre valeurs possibles pour x_2 apparaissent. C'est-à-dire que peu importe la valeur que prend x_2 alors x_3 est toujours différent de 1 à ce stade de la recherche.

En considérons cela, nous pouvons minimiser indirectement les *increasing-nogoods* selon les scénarios de recherche, voire directement si toutes les valeurs de l'alpha

d'un groupe apparaissent.

Algorithme 6 : checkAlphaAndNegDec()

```
1 foreach x ∈ alphaSet do
2     if diffValuesAlpha(x) ⊇ dom(x) then
3         Δ = ⋂{δ_i ∈ ng_j | j ∈ alphaToNG(x)};
4         foreach δ ∈ Δ do
5             satisfy δ;
6         end
7     end
8 end
```

Soit `alphaSet`, un ensemble dynamique des différentes variables pointées par alpha dans chaque *IncNG*, l'algorithme 6 permet de rechercher les décisions négatives communes dans les groupes qui portent sur la même variable pointée par alpha (la valeur peut être différente). Dans ce but, nous utilisons deux fonctions :

- `diffValuesAlpha(x)` qui permet de retourner l'ensemble de valeurs différentes prises pour une même décision x pointée par alpha dans la base globale d'*increasing-nogoods*,
- `alphaToNG` qui utilise un ensemble de correspondances entre la variable de l'alpha et les *increasing-nogoods* où elle apparaît.

Ces fonctions permettent de construire l'ensemble Δ qui contient les décisions négatives communes à un groupe où toutes les valeurs d'une même variable pointée par alpha apparaissent. L'algorithme 6 a une complexité dans le pire cas en $O(ndg^2)$ où n est le nombre de variables total, d la taille du plus grand domaine et g le nombre d'*increasing-nogoods* dans la base.

4 Expérimentations

Les expériences présentées ci-après sont réalisées sur des machines équipées d'un processeur *Intel Xeon X5550* cadencé à 2,67 GHz ainsi que de 8 Go de mémoire, avec un *timeout* réglé à 15 minutes. Nous avons sélectionné un panel représentatif de 3744 problèmes provenant des compétitions CP de 2006 et de 2008. Les résultats sont présentés sous forme de tableaux où les familles sont regroupées.

Pour ces expérimentations, nous avons utilisé une version simplifiée du solveur *rclCSP* présenté dans [6] utilisant l'heuristique *dom/wdeg* [1]. Étant donnée la complexité de l'algorithme 6 et le fait que l'algorithme 4 soit subsumé par l'algorithme 5, nous avons choisi de ne reporter que ce dernier. Dans la suite, il est référencé par le symbole $\alpha_\Leftrightarrow$ dans les tableaux.

Afin de montrer la corrélation entre le nombre et la taille des *nogoods*, nous utilisons différentes stratégies de redémarrage basées sur le nombre de conflits. Celles-ci sont, soit basées sur la suite de Luby [14] (1,1,2,1,1,2,4,...), soit sur une suite géométrique. Pour ces expérimentations, nous avons utilisé les politiques suivantes :

- P50 - suite géométrique de premier terme 50 et de raison 1.5 ;
- P10 - suite géométrique de premier terme 10 et de raison 1.1 ;
- L100 - suite de luby dont les termes sont multipliés par 100.

Le tableau 1 reporte une comparaison entre les *increasing-nogoods* avec et sans $\alpha_\Leftrightarrow$ sur la famille de problème *QCP-20*. Dans ce tableau, nous reportons le nombre total de *nld-nogoods* apparaissant dans les *increasing-nogoods* (*#nld*), le nombre total de suppressions induites par les *increasing-nogoods* (*#del*), en parenthèse apparaît le nombre de suppressions dues à $\alpha_\Leftrightarrow$ (ce nombre est inclus dans le total). Ce tableau reporte aussi le nombre de conflits imputés par la révision des *increasing-nogoods* (*#fail*), ainsi que le temps de résolution, exprimé en secondes (*cpu*). Sur ce tableau, nous pouvons remarquer que sur cette famille d'instances l'ajout d'$\alpha_\Leftrightarrow$ permet de résoudre une instance de plus. Nous pouvons aussi remarquer que de nombreuses suppressions de valeurs sont induites par l'ajout de notre méthode. Finalement, nous observons que pour la majorité des problèmes de cette famille, le nombre de conflits obtenus dans les *increasing-nogoods* est plus faible. Cela peut être s'expliquer par le fait que les conflits sont détectés plus tôt et donc que l'arbre de recherche est plus petit.

Le tableau 2 permet d'évaluer l'impact de la stratégie de redémarrage sur les performances des méthodes considérées. Pour cela nous avons comparé trois méthodes, *increasing-nogoods* avec et sans $\alpha_\Leftrightarrow$ ainsi que les *nld-nogoods*. Cette table reporte, pour l'ensemble des problèmes considérés dans cette expérience, le nom de la famille considérée (*Family name*) ainsi que le nombre de problèmes dans celle-ci (*#prob*). Pour chacune des méthodes, nous reportons la stratégie de redémarrage considérée ainsi que le nombre d'instances résolues (*#sol*) et le temps moyen de résolution (*cpu*). Comme nous pouvons voir sur ce tableau, une stratégie de redémarrage moins agressive est souhaitable lorsque nous considérons les *nld-nogoods* ou les *increasing-nogoods* seuls. Dans le cas de notre méthode $\alpha_\Leftrightarrow$, une stratégie plus agressive permet d'utiliser pleinement la puissance des *increasing-nogoods*. Cela peut en partie s'expliquer par le fait d'une base plus importante de *nogoods* permet d'induire un

nombre plus important de suppressions.

Lors de nos expérimentations nous avons remarqué que plus nous utilisons une politique de redémarrage agressive plus les combinaisons sont efficaces d'un point de vue suppressions mais la complexité [1], dépendant de la taille de la base, augmente. Nous avons donc lancé quelques simulations pour mesurer l'impact en temps de calcul sur une politique relativement agressive, à savoir *luby* $\times$ *10*. Nous avons empêché les algorithmes relatifs aux *increasing-nogoods* ainsi qu'à $\alpha_{\Leftrightarrow}$ de faire le moindre changement sur le réseau mais ils continuent à être appelés normalement. Le résultat est le suivant sur l'instance scen11-f5 le temps de résolution est de 120 secondes dont 85 de calcul d'équivalence d'alpha et 2,5 secondes de gestion des *increasing-nogoods*. Cette même instance est résolue en 17,45 secondes (dont 3,9 secondes de combinaisons) si nous activons les suppressions. De ces résultats, nous envisageons quelques pistes d'amélioration dans la section suivante.

5 Travaux futurs

À la vue des résultats encourageants nous avons décidé d'améliorer les algorithmes présentés précédemment grâce à diverses méthodes. Dans cette section nous allons présenter une première méthode, à savoir un système de sentinelles.

Les sentinelles peuvent s'apparenter à un système de *1-watch* appliqué à des domaines relatifs à des *increasing-nogoods*. Nous allons pouvoir les utiliser dans deux des trois méthodes de combinaison présentées, la combinaison de décisions négatives (présentée section 3.1) ainsi que la combinaison par équivalence de décisions négatives (présentée section 3.3).

Dans le cas de la combinaison de décisions négatives, la théorie est la suivante : dans les décisions négatives contenues entre alpha et beta par un *IncNG*, c'est-à-dire là où nous espérons avoir des valeurs supprimées, nous pouvons ajouter un certain nombre de sentinelles. Une sentinelle marque une valeur que nous ne regardons pas normalement dans le domaine de la variable associée. Tant que celle-ci existe nos suppressions ne créeront pas de conflits. De plus, grâce à ce système de sûreté de domaine nous n'avons plus à nous soucier des valeurs négatives lors du filtrage d'un *IncNG* dans le cas où δ_α n'est pas assigné, nous pouvons, par conséquent, supprimer les lignes 13 à 21 de l'algorithme 2.

Nous proposons de mettre en place un système de sentinelles qui, associé à chaque *IncNG*, regarde si une valeur différente de celle apparaissant dans le *IncNG* existe. Tant qu'il reste un élément autre que ceux apparaissant dans le *IncNG*, les suppressions pourront se faire sans conflit si δ_α venait à être satisfaite. Si la sentinelle venait à être supprimée sans possibilité d'en trouver une nouvelle, il faudrait alors supprimer la valeur pointée par α. Cela peut se faire en utilisant soit des *1-watch*, soit de manière paresseuse en vérifiant la taille des domaines concernés par rapport aux nombres de valeurs qui apparaissent négativement dans le *IncNG*, cette dernière étant beaucoup moins précise.

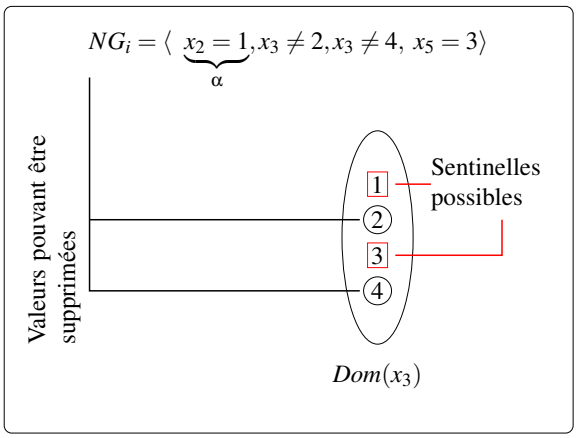


FIGURE 2 – Exemple de fonctionnement des sentinelles

Nous pouvons voir, dans la figure 2, qu'il est possible de choisir deux sentinelles, les valeurs 1 et 3 du domaine de x_3. Tant qu'au moins une de ces valeurs est dans le domaine, nous n'avons pas besoin de falsifier la décision pointée par alpha.

Dans le cas de la combinaison par équivalence de décisions négatives, nous pouvons utiliser une sentinelle pour détecter lorsque la taille du domaine de l'alpha commun (même variable, valeur différente) arrive à la taille du sous-groupe.

Toujours dans le but d'étendre les interactions entre *nogoods* nous proposons une méthode alternative, voire complémentaire, aux combinaisons proposées dans cet article. Nous considérons toujours des groupes avec décisions négatives équivalentes mais avec des alphas quelconques. Grâce à ceux-ci, nous générons des *nld-nogoods* réduits composés des prémices, c'est-à-dire les décisions positives avant l'alpha courant, ainsi que des alphas. Cette méthode peut se généraliser aux combinaisons par équivalence de décisions négatives ainsi qu'aux combinaisons par équivalence d'alpha.

1. $O(nsg)$ où n est le nombre de variables, s la taille du plus grand *IncNG* et g la taille de la base d'*IncNG*.

P10	IncNG + $\alpha_{\Leftrightarrow}$				IncNG			
	cpu	#nld	#del (dont $\alpha_{\Leftrightarrow}$)	#fail	cpu	#nld	#del	#fail
qcp-20-187-0	222,61	671	38	1	**208,05**	671	38	1
qcp-20-187-1	176,41	806	11033(708)	289	**31,75**	569	2287	107
qcp-20-187-2	timeout	1019	57232(2088)	6665	timeout	1007	54728	6980
qcp-20-187-3	timeout	770	33255(2027)	1881	timeout	844	53057	5043
qcp-20-187-4	timeout	818	39867(2548)	1905	timeout	862	47013	3692
qcp-20-187-5	**0,7**	121	15	0	**0,7**	121	15	0
qcp-20-187-6	634,87	835	5177(1)	573	**597,85**	835	5177	573
qcp-20-187-7	**258,52**	679	1267(17)	40	timeout	847	1582	90
qcp-20-187-8	**32,94**	462	807(116)	6	205,33	667	5286	350
qcp-20-187-9	**15,54**	396	72	2	15,62	396	72	2
qcp-20-187-10	timeout	921	21082(667)	1362	timeout	892	43964	3780
qcp-20-187-11	**0,27**	63	44(22)	3	0,3	62	38	8
qcp-20-187-12	timeout	880	21178(540)	1524	timeout	926	34894	6362
qcp-20-187-13	timeout	863	102101(2906)	8127	timeout	946	191009	33687
qcp-20-187-14	timeout	983	137251(13389)	6261	timeout	930	30693	1819

TABLE 1 – Résultats expérimentaux détaillés sur quelques problèmes.

6 Conclusion

Cet article a mis en évidence qu'il était possible de combiner les informations entre les *increasing-nogoods* et la structure du problème afin d'augmenter significativement le pouvoir de filtrage. Plus précisément, nous avons d'abord proposé une approche qui, étant donné un *nogood*, prend en considération les informations relatives aux domaines des variables afin d'identifier de nouvelles valeurs à supprimer. Ensuite, nous avons montré qu'il est possible d'étendre ce processus à la base entière d'*increasing-nogoods* de deux manières, soit en fonction de leurs décisions positives, soit en fonction de leurs décisions négatives.

Nous avons montré sur un large panel d'instances que ces algorithmes de filtrage permettent de supprimer de nombreuses valeurs lors du processus de propagation. Cependant, l'utilisation de ces algorithmes a deux inconvénients. Le premier est que le fait de détecter les conflits en amont à un impact sur l'heuristique de choix de variables. En effet, puisque l'heuristique utilisée est ajustée en fonction des conflits, les éviter ne permet pas d'augmenter la pondération des contraintes, et donc de choisir la meilleure variable. L'autre inconvénient est la complexité des algorithmes présentés, qui dépend de la taille de la base d'*increasing-nogoods*. Comme énoncé dans la section 5 cette situation peut être améliorée en considérant des mécanismes de mise à jour basés sur l'utilisation d'un système de sentinelles ou d'ensembles persistant *backtrack-ready*.

Références

[1] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.

[2] R. Dechter. Enhancement schemes for constraint processing : backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41 :273–312, 1990.

[3] T. Feydy and P. Stuckey. Lazy clause generation reengineered. In *Proceedings of CP'09*, pages 352–366, 2009.

[4] D. Frost and R. Dechter. Dead-end driven learning. In *Proceedings of AAAI'94*, pages 294–300, 1994.

[5] M.L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1 :25–46, 1993.

[6] E. Gregoire, J.M. Lagniez, and B. Mazure. A CSP solver focusing on fac variables. In *Proceedings of CP'11*, pages 493–507, 2011.

[7] N. Jussien, R. Debruyne, and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Proceedings of CP'00*, pages 249–261, 2000.

[8] G. Katsirelos and F. Bacchus. Unrestricted nogood recording in CSP search. In *Proceedings of CP'03*, pages 873–877, 2003.

[9] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Nogood recording from restarts. In *Proceedings of IJCAI'07*, pages 131–136, 2007.

[10] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Recording and minimizing nogoods from restarts. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 1 :147–167, 2007.

[11] C. Lecoutre and S. Tabary. Symmetry-reinforced nogood recording from restarts. In *Proceedings of SymCon'11*, pages 13–27, Perugia, Italy, 2011.

[12] J. H. M. Lee, C. Schulte, and Z. Zhu. Increasing nogoods in restart-based search. In *Proceedings of AAAI'16*, pages 3426–3433, 2016.

[13] J. H. M. Lee and Z. Zhu. An increasing-nogoods global constraint for symmetry breaking during search. In *Proceedings of CP'14*, pages 465–480, 2014.

[14] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Inf. Process. Lett.*, 47(4) :173–180, 1993.

[15] T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *International Journal of Artificial Intelligence Tools*, 3(2) :187–207, 1994.

		NLD						IncNG						IncNG + $\alpha_{\Leftrightarrow}$					
		L100		P50		P10		L100		P50		P10		L100		P50		P10	
Family name	#prob	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu
allIntervalSeries	25	10	15,77	**12**	**45,12**	11	15,48	10	13,45	11	7,31	11	19,39	10	13,93	11	7,34	11	35,81
bddLarge	35	35	60,3	35	52,26	35	128,88	**35**	**30,49**	35	30,54	35	33,15	35	30,56	35	30,74	35	33,16
bddSmall	35	35	109,52	35	43,35	35	113,69	35	25,4	35	25,2	35	25,65	35	25,3	**35**	**24,99**	35	25,48
bibd10-11	6	3	3,88	3	232,16	1	0,21	2	0,85	**4**	**175,44**	2	2,43	2	1,27	4	178,82	2	2,31
bibd12-13	7	**5**	10,09	3	1,78	4	1,97	4	4,87	4	10,63	3	5,29	4	4,39	4	11,36	3	5,12
bibd8	7	3	166,18	3	95,17	3	58,05	3	88,95	3	61,82	**3**	41,97	3	180,02	3	63,5	3	44,52
bibd9	10	10	1,46	10	2,47	10	4,53	10	1,41	10	2,41	8	4,3	**10**	**1,37**	10	2,2	8	4,31
bibdVariousK	29	23	7,22	23	60,36	22	28,68	**25**	**31,27**	21	4,44	23	6,47	24	12,78	22	42,04	23	7,53
bmc	24	24	8,19	24	14,32	24	30,47	24	7,99	24	15,11	24	22,57	**24**	**7,94**	24	16,06	24	20,82
chessboardColoration	20	15	6,76	15	7,82	15	21,81	15	3,66	15	5,67	15	6,4	**15**	**3,65**	15	5,67	15	5,76
coloring	22	21	0,58	22	4,37	22	8,51	22	9,44	**22**	**3,48**	22	3,89	22	16,64	22	4,28	22	4,85
compet08	16	9	85,22	9	107,12	9	132,69	8	70,17	**10**	**151,98**	9	81,72	9	157,09	10	157,34	9	87,2
costasArray	11	7	52,19	9	78,21	9	133,82	8	104,29	8	71,13	**9**	**62,42**	8	96,51	8	68,77	9	64,41
dag-half	25	1	898,31	6	631,48	0	NaN	17	412,64	16	370,99	17	385,25	**18**	**408,44**	16	406,4	17	373,83
dag-rand	25	24	448,24	25	163,81	25	282,27	25	136,01	25	132,93	25	131,57	25	134,59	25	132,27	**25**	**129,97**
dubois	13	0	NaN	6	275,72	4	261,5	4	246,52	**6**	**155,64**	6	174,93	3	319,69	6	166,99	6	214,52
fapp11-15	55	41	80,75	41	41,49	42	46,44	43	62,34	42	52,14	42	128,43	**43**	**61,48**	42	53,05	42	37,93
frb30-15	10	10	0,91	**10**	**0,68**	10	1,02	10	0,87	10	0,78	10	0,84	10	0,84	10	0,83	10	0,87
frb35-17	10	10	5,22	10	2,75	10	3,13	10	1,95	**10**	**1,81**	10	1,97	10	2,24	10	1,91	10	2,05
frb40-19	10	10	210,05	10	31,11	10	39,64	10	33,4	10	29,61	10	29,57	10	33,87	**10**	**27,25**	10	29,02
frb45-21	10	1	398,13	10	248,76	9	265,95	9	222,56	9	229,33	**10**	**233,17**	9	243,82	9	251,02	10	283,14
frb50-23	10	0	NaN	**2**	**457,53**	1	456,36	0	NaN	1	844,67	1	794,87	1	454,4	1	850,63	1	858,43
golombRulerArity3	14	9	57,76	11	131,92	10	61,76	10	50,99	10	51,37	10	51,36	10	56,17	**11**	**123,79**	10	49,83
golombRulerArity4	14	8	78,86	**10**	**55,28**	10	79,38	10	62,02	10	58,74	10	59,17	10	62,57	10	58,96	10	59,15
graphColoring	458	185	19,15	196	25,86	196	30,84	196	27,81	**198**	**32,11**	197	27,81	195	30,54	197	28,66	197	28,68
lexHerald	47	39	33,22	41	39,32	41	41,51	**44**	**55,01**	41	66,24	43	59,36	44	69,71	40	33,47	43	63,28
lexVg	63	62	50,3	63	32,6	63	44,45	63	30,74	**63**	**27,55**	63	30,74	63	29,04	63	28,38	63	30,58
magicSquare	18	5	3,01	7	114,26	6	6,51	6	103,33	6	59,44	7	89,88	6	105,37	6	59,93	**7**	**89,02**
marc	10	**10**	**8,1**	10	8,11	10	8,15	10	8,5	10	8,23	10	8,35	10	8,41	10	8,32	10	8,22
mknap	6	4	78,03	4	39,8	4	91,26	4	23,06	**4**	**22,67**	4	23,2	4	22,72	4	22,74	4	22,91
nengfa	10	9	123,78	8	59,26	9	79,37	9	85,85	9	89,78	**9**	**61,02**	9	89,71	9	79,23	9	83,71
ogdHerald	50	**50**	**2,4**	50	2,85	50	4,61	50	2,56	50	2,58	50	3,07	50	2,48	50	2,6	50	3,08
ogdPuzzle	22	22	0,76	22	0,76	22	0,78	22	0,84	22	0,83	22	0,71	22	0,79	**22**	**0,7**	22	0,75
ogdVg	65	34	73,9	39	93,04	35	81,07	**40**	**74,33**	39	55,27	39	63,24	40	77,1	40	81,11	40	81,09
ortholatin	9	3	31,98	3	15,46	**4**	**102,44**	3	54,38	3	13,77	3	34,57	3	54,27	3	13,7	3	33,37
os-taillard-10	30	20	33,42	17	42,09	20	31,87	20	32,34	18	65,09	**20**	**30,4**	20	32,69	18	64,08	20	30,68
os-taillard-15	30	21	7,49	20	7,22	21	9,39	21	6,94	20	6,68	21	8,84	21	7,01	**20**	**6,86**	21	8,86
os-taillard-20	30	22	24,23	21	26,03	22	29,06	22	22,18	21	24,66	22	28,69	**22**	**20,47**	21	24	22	28,65
os-taillard-4	30	30	0,78	30	0,67	30	0,79	30	0,59	30	0,69	30	4,07	30	0,69	**30**	**0,58**	30	0,7
os-taillard-5	30	28	49,5	29	39,24	29	47,16	29	37,04	28	28,33	**30**	**27,97**	30	57,02	29	57,61	30	28,44
os-taillard-7	30	16	72,55	17	72,95	**19**	**85,25**	16	65,69	17	54,9	18	133,96	17	73,57	17	55,04	18	134,68
pseudoGLB	384	132	310,84	187	135,54	152	296,18	**198**	**73,22**	190	66,81	186	67,01	189	72,04	189	64,38	186	77,17
pseudo	486	257	36,88	268	29,1	258	35,12	**281**	**36,13**	272	30,34	274	26,8	278	32,09	272	31,46	274	30,61
QCP-15	15	15	9,2	15	2,85	15	4,79	15	2,04	15	2,57	**15**	**1,77**	15	2,35	15	2,61	15	2,31
QCP-20	15	5	137,33	7	49,31	6	93,76	8	208,02	7	133,62	7	151,93	6	55,57	7	164,33	**8**	**168,18**
QG3	7	5	2,12	5	1,71	5	2,45	5	1,46	**5**	**1,15**	5	1,7	5	1,6	5	1,33	5	1,67
QG4	7	5	2,78	5	2	5	2,88	5	1,95	**5**	**1,53**	5	2,36	5	1,58	5	1,54	5	2,31
QG5	7	6	75,46	6	15,97	6	22,21	6	13,24	6	12,6	**6**	**12,52**	6	13,96	6	12,69	6	12,62
QG6	7	6	6,97	6	3,28	6	6,04	6	2,06	6	2,12	6	1,12	6	2,09	6	1,97	**6**	**1,1**
QG7	7	4	15,62	5	7,37	5	11,91	**5**	**4,7**	5	53,16	5	11,92	5	4,76	5	54,03	5	12,2
queenAttacking	10	5	25,4	5	4,11	5	2,42	5	2,63	5	50,82	5	3,99	**5**	**2,37**	5	50,75	5	3,74
queensKnights	18	16	70,54	17	99,89	17	91,62	16	57,53	**18**	**105,63**	18	116,45	16	58,46	18	105,68	18	130,59
QWH-20	10	9	50,02	10	101,23	10	80,04	10	88,01	**10**	**40,99**	10	48,26	9	22,38	10	68,25	10	49,01
QWH-25	10	0	NaN	1	590,09	1	875,89	1	794,82	**3**	**310,66**	0	NaN	1	838,62	2	265,2	2	458,58
rand-10-20-10	20	20	2,78	20	2,81	20	2,97	**20**	**2,64**	20	2,84	20	2,94	20	2,66	20	2,82	20	2,89
rand-2-23	10	10	103,09	10	33,17	10	38,01	10	29,65	**10**	**26,72**	10	32,53	10	36,07	10	29,35	10	28,17
rand-2-24	10	10	405,85	10	82,45	10	79,37	10	81,45	10	69,53	**10**	**67,9**	10	98,83	10	71,09	10	71,03
rand-2-25	10	3	515,25	10	155,91	10	173,81	10	180,25	10	141,47	10	149,84	10	280,6	**10**	**125,12**	10	138,37
rand-2-26	10	1	55,54	10	372,33	10	397,49	10	401,18	10	339,67	10	346,05	5	161,03	**10**	**320,91**	10	392,58
rand-2-27	10	0	NaN	3	357,2	4	275,15	4	435,21	4	295,76	**4**	**236,98**	3	295,63	4	459,59	4	341,94
rand-2-30-15-fcd	50	50	0,83	50	0,72	50	0,88	50	0,74	**50**	**0,7**	50	0,77	50	0,75	50	0,79	50	0,8
rand-2-30-15	50	50	1,13	50	1	50	1,06	**50**	**0,94**	50	0,98	50	1	50	1,04	50	0,99	50	1
rand-2-40-19-fcd	50	48	194,81	50	26,88	50	41,12	50	29,1	**50**	**25,52**	50	29,75	50	32,4	50	31,95	50	29,92
rand-2-40-19	50	40	296,87	50	66,82	50	81,37	50	63,86	50	61,55	**50**	**61,36**	50	71,01	50	64,06	50	62,17
rand-2-50-23-fcd	50	0	NaN	14	505,07	5	462,74	12	524,45	15	457,11	12	470,7	9	381,09	15	522,33	**15**	**456,83**
rand-2-50-23	50	0	NaN	12	514,2	4	685,69	12	473,67	12	597,81	9	539,14	8	519,13	**13**	**568,28**	9	460,74
rand-3-20-20-fcd	50	50	56,15	50	28,07	50	34,29	50	28,55	**50**	**22,67**	50	26,77	50	30,42	50	23,41	50	24,69
rand-3-20-20	50	50	104,04	50	51,46	50	56,07	50	47,14	50	45,94	50	47,82	50	48,9	**50**	**44,71**	50	47,33
rand-3-24-24-fcd	50	10	413,14	26	272,73	23	326,17	28	314,43	26	297,64	27	314,23	**29**	**272,14**	26	271,52	28	301,76
rand-3-24-24	50	6	536,52	16	467,63	14	415,31	15	425,36	14	367,36	16	401,12	16	405,91	15	478,29	**18**	**390,44**
rlfapScens11	12	10	84,9	11	103,52	11	112,19	11	98,85	11	101,48	11	103,31	10	39,88	**11**	**98,48**	11	112,5
schurrLemma	10	8	65,17	8	76,64	8	92,92	9	71,41	9	158,85	8	77,14	**9**	**71,1**	8	78,43	9	120,94
super-jobShop	46	33	12,35	35	31,83	34	15,24	36	36,12	35	24,63	33	6,68	**36**	**26,86**	34	20,62	35	45,39
tdsp	42	5	30,47	5	38,76	5	171,76	5	2,12	5	73,06	5	5,43	5	1,71	6	103,54	**6**	**27,42**
tightness0,1	100	100	11,33	100	4,6	100	6,13	100	4,42	**100**	**4,07**	100	4,28	100	4,54	100	4,17	100	4,62
tightness0,2	100	100	17,92	100	6,49	100	8,67	100	6,07	**100**	**5,7**	100	6,17	100	6,57	100	6,01	100	6,18
tightness0,35	100	100	13,27	100	6,18	100	7,18	100	5,53	**100**	**5,33**	100	5,59	100	5,88	100	5,44	100	5,68
tightness0,5	100	100	16,68	100	7,29	100	8,21	100	6,87	100	7,19	100	7,1	100	7,24	**100**	**6,86**	100	7
tightness0,65	100	100	7,23	100	5,05	100	5,41	**100**	**4,55**	100	4,74	100	4,63	100	4,86	100	4,82	100	4,66
tightness0,8	100	100	6,27	100	4,67	100	5,35	100	5,01	**100**	**4,64**	100	4,75	100	4,68	100	4,69	100	4,78
tightness0,9	100	100	7,83	100	6,86	100	7,08	100	7,12	100	7,4	100	7,35	100	7,11	100	7,47	**100**	**6,66**
TOTAL	3744	2443		2648		2571		2687		2668		2665		2662		2669		2679	

TABLE 2 – Résultats expérimentaux par familles des trois méthodes : NLD, IncNG et IncNG + $\alpha_{\Leftrightarrow}$ sur différentes politiques de redémarrage.

Extension de Compact-Table aux tables négatives et concises

Hélène Verhaeghe[1] * **Christophe Lecoutre**[2] **Pierre Schaus**[1]

[1]UCLouvain, ICTEAM, Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgique
[2]CRIL-CNRS UMR 8188, Université d'Artois, F-62307 Lens, France
[1]{*prenom.nom*}@*uclouvain.be* [2]*lecoutre@cril.fr*

Résumé

Ces dernières années, plusieurs algorithmes pour la contrainte table ont été proposés pour assurer la propriété de cohérence d'arc généralisée (GAC). Compact-Table (CT) [1] est un algorithme récent de l'état-de-l'art, que nous avons étendu dans notre article [4], intitulé "Extending Compact-Table to Negative and Short Tables" et publié à AAAI-17, aux tables concises (contenant des supports courts) et aux tables négatives (contenant des conflits).

1 Introduction

La contrainte table, ou contrainte en extension, exprime explicitement pour les variables impliquées, soit les combinaisons de valeurs acceptées (*supports*), soit les combinaisons de valeurs rejetées (*conflits*). En théorie, toute contrainte peut se reformuler sous forme de contrainte table, c'est l'une des raisons pour lesquelles ce type de contrainte est très important en programmation par contraintes.

Beaucoup d'efforts ont été consentis au développement d'algorithmes de filtrage pour les contraintes tables, le plus efficace ayant été demontré être Compact-Table [1], proposé en 2016. L'un des inconvénients avec les tables est que leur taille, et donc l'utilisation mémoire requise, peut potentiellement augmenter de manière exponentielle par rapport à l'arité. Pour pallier ce problème, plusieurs méthodes de compressions ont été proposées : les *tries*, les Diagrammes de Décision Multi-valeurs (*MDDs*) et les Automates Finis Déterministes (*DFAs*), qui sont des représentations pouvant faciliter le processus de filtrage.

*Papier doctorant : Hélène Verhaeghe[1] est auteur principal.

D'autre approches, basées sur le concept de produit cartésien, ont également été envisagées : tuples compressés (*compressed tuples*), supports courts (*short support*), tables découpées (*sliced tables*), tables intelligentes (*smart tables*),...

Dans cet article, nous étendons l'algorithme Compact-Table, initialement introduit pour les tables positives sans aucune compression [1], afin de pouvoir gérer :

— les tables négatives (i.e. les tables de conflits) ;
— et/ou les tables concises, i.e. les tables avec supports courts qui sont des tuples contenant la valeur universelle * représentant n'importe quelle valeur pour une variable.

2 Gérer les tables concises : CT*

De manière intéressante, nous avons pu observer que CT peut être facilement adapté pour gérer les tables positives avec supports courts et cela sans dégradation de la complexité : elle reste en $\mathcal{O}(rd\frac{t}{w})$, où r est l'arité, d la taille du domaine le plus grand, t le nombre de tuples dans la table concise (contenant les *) et w la taille d'un mot processeur (e.g. $w = 64$).

La modification se caractérise par une nouvelle version de la règle de mise à jour de la table des tuples encore acceptables : pour une variable donnée x, le tuple τ est retiré de l'ensemble des tuples encore acceptables si $\tau(x) \neq *$ et $\tau(x) \notin D_x$, où D_x est le domaine de la variable x.

Les résultats obtenus, sur 600 instances variées et aléatoirement générées, avec CT*, ShortSTR2 (extension de STR2 gérant des supports courts [2]), CT et STR2 (pour ces deux derniers, les tables en entrée sont les tables équivalentes décompressées), sont

visibles sur la figure 1, sous forme de profiles de performance.

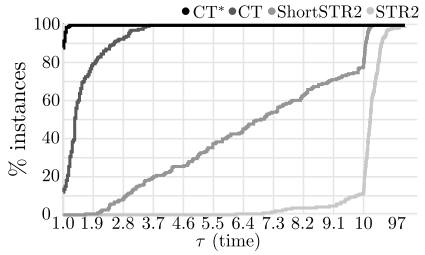


FIGURE 1 – Résultats sur tables positives concises

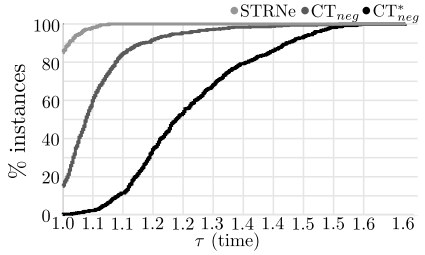


FIGURE 2 – Résultats sur tables négatives (instances avec solutions multiples)

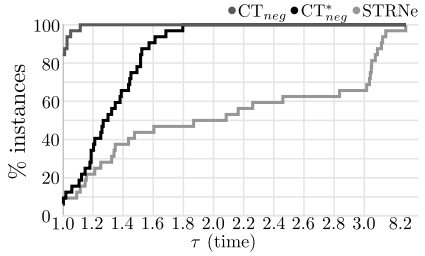


FIGURE 3 – Résultats sur tables négatives (instances sans solutions)

3 Gérer les tables négatives : CT$_{neg}$

Le changement se situe ici lors du processus de filtrage. Pour déterminer si un litéral (x,v) doit être conservé, il faut vérifier qu'il existe au moins un tuple valide, considérant l'état courant des domaines, qui ne corresponde pas à un conflit. Cette vérification peut être réalisée en comptant le nombre de conflits toujours valides contenant ce litéral. On compare cette valeur au produit des tailles courantes des domaines de toutes les variables, excepté celle du litéral. Une égalité entre ces deux nombres implique qu'il n'y a aucun tuple réalisable avec le litéral qui ne soit pas un conflit.

La complexité devient $\mathcal{O}(rd\frac{t}{w}k)$, k étant la complexité pour compter le nombre de 1 présents dans un mot processeur. Dans notre implémentation, avec l'utilisation de Long.bitCount, $k = log(w)$, mais sur certaines architecture, nous avons même $k = 1$.

Les résultats obtenus sur des instances variées et aléatoirement générées sont donnés par les figures 2 et 3. On peut y constater que nous gagnons vraiment en efficacité par rapport à STRNe [3] lorsque nous traitons des instances incohérentes (i.e. sans solutions).

4 Gérer les tables négatives et concises : CT$^*_{neg}$

Les modifications faites pour gérer les conflits avec des supports courts se basent sur l'agrégation des deux idées à la base de CT* et CT$_{neg}$ moyennant l'hypothèse de ne pas avoir de tuples se superposant. Cette hypothèse est nécessaire pour garder un comptage trivial des tuples.

5 Conclusion

Nous avons proposé une extension de Compact-Table qui exploite l'efficacité des opérations bit à bit. Cela permet d'opérer, de manière compétitive, sur les tables concises (CT*), les tables négatives (CT$_{neg}$) et les tables négatives concises (CT$^*_{neg}$). Nous pensons qu'elle sera implantée dans les solvers car les tables consises et/ou négatives vont devenir de plus en plus populaires, apportant à l'utilisateur une certaine facilité de modélisation. Pour plus de détails concernant l'implémentation et les résultats, n'hésitez pas à lire l'article originel.

Références

[1] J. Demeulenaere, R. Hartert, C. Lecoutre, G. Perez, L. Perron, J.-C. Régin, and P. Schaus. Compact-table : efficiently filtering table constraints with reversible sparse bit-sets. In *Proceedings of CP'16*, pages 207–223, 2016.

[2] C. Jefferson and P. Nightingale. Extending simple tabular reduction with short supports. In *Proceedings of IJCAI'13*, pages 573–579, 2013.

[3] Hongbo Li, Yanchun Liang, Jinsong Guo, and Zhanshan Li. Making simple tabular reduction works on negative table constraints. In *Proceedings of AAAI'13*, pages 1629–1630, 2013.

[4] Hélène Verhaeghe, Christophe Lecoutre, and Pierre Schaus. Extending compact-table to negative and short tables. In *Proceedings of AAAI'17*, 2017.

Une approche basée sur SAT pour le problème de satisfiabilité en logique modale S5

Thomas Caridroit Jean-Marie Lagniez Daniel Le Berre
Tiago de Lima Valentin Montmirail *

CRIL, Univ. Artois et CNRS, F62300 Lens, France
{caridroit,lagniez,leberre,delima,montmirail}@cril.fr

Résumé

Nous présentons une approche basée sur SAT pour résoudre le problème de satisfiabilité en logique modale S5. Ce problème étant NP-complet, la traduction en SAT n'est pas une surprise. Notre contribution est de réduire considérablement le nombre de variables propositionnelles ainsi que de clauses nécessaires pour coder le problème. Nous présentons dans un premier temps une propriété syntaxique appelée *diamond degree*. Nous montrons que la taille d'un modèle S5 satisfaisant une formule ϕ peut être bornée par ce *diamond degree*. Une telle mesure peut ainsi être utilisée comme borne supérieure pour générer un codage SAT pour la satisfiabilité S5 de cette formule. Nous proposons ensuite un système de *caching* qui nous permet de réduire la taille de la formule propositionnelle. Nous avons implémenté une approche générique basée sur SAT dans le solveur S52SAT. Celle-ci nous permet de comparer expérimentalement notre nouvelle borne supérieure à celle connue antérieurement, c'est-à-dire le nombre de modalités de ϕ, ainsi que d'évaluer l'effet de notre technique de *caching*. Nous comparons également notre solveur avec des solveurs existants en logique modale S5. L'approche proposée surpasse les précédentes sur les benchmarks utilisés. Ces résultats prometteurs ouvrent des perspectives de recherche intéressantes pour la résolution pratique d'autres logiques modales (par exemple K, KT, S4).

Abstract

We present a SAT-based approach for solving the modal logic S5-satisfiability problem. That problem being NP-complete, the translation into SAT is not a surprise. Our contribution is to greatly reduce the number of propositional variables and clauses required to encode the problem. We first present a syntactic property called diamond degree. We show that the size of an S5-model satisfying a formula ϕ can be bounded by its diamond degree. Such a measure can thus be used as an upper bound for generating a SAT encoding for the S5-satisfiability of that formula. We also propose a lightweight caching system which allows us to further reduce the size of the propositional formula. We implemented a generic SAT-based approach within the modal logic S5 solver S52SAT. It allowed us to compare experimentally our new upper-bound against the previously known one, i.e. the number of modalities of ϕ and to evaluate the effect of our caching technique. We also compared our solver against existing modal logic S5 solvers. The proposed approach outperforms previous ones on the benchmarks used. These promising results open interesting research directions for practical reasoning in other modal logics (e.g. K, KT, S4).

Introduction

Au cours des vingt dernières années, les logiques modales ont été utilisées dans divers domaines de l'intelligence artificielle comme la vérification formelle [10], la théorie des jeux [25], la théorie des bases de données [11] et l'informatique distribuée [38]. Plus récemment, la logique modale S5 a été utilisée pour la planification contingente [29] et la compilation de connaissances [5]. Pour cette raison, le raisonnement automatisé en logiques modales a été largement étudié (par exemple [28, 34, 35]). Ladner [9, 24] a montré que le problème de satisfiabilité pour plusieurs logiques modales comprenant K, KT et S4 est PSPACE-complet alors qu'il est NP-Complet pour la logique S5 (voir [21] pour plus de détails). Puisque les solveurs SAT sont devenus des oracles NP assez efficaces pour de nombreux problèmes, nous nous intéressons à étudier l'encodage SAT pour le problème S5-SAT d'un point de vue pratique. Utiliser un oracle SAT dans le cadre de la logique modale n'est pas nouveau : un aperçu complet des travaux antérieurs peut être trouvé dans [33]. Cependant,

*Papier doctorant : Valentin Montmirail est auteur principal

la plupart d'entre eux s'attaquent à la satisfiabilité de la logique modale K. *SAT [13, 14, 15, 17] utilise un oracle SAT pour décider de la satisfaction de 8 logiques modales différentes, y compris K, mais pas S5. Dans le même esprit que notre travail, une traduction de la logique modale K vers SAT a été proposée dans Km2SAT [18, 34]. Plus récemment, le solveur InKreSAT [23] a proposé un système innovant basé sur SAT où le solveur SAT conduit le développement d'une méthode des tableaux. Sur un plan théorique, une approche basée sur SMT (Satisfiability Modulo Theory) [2] a également été proposée. Aucune de ces méthodes n'est applicable à la logique modale S5.

Le nombre de variables nécessaires pour réduire S5-SAT à SAT dépend d'une borne supérieure du nombre de mondes possibles à considérer dans le modèle S5. Minimiser cette borne supérieure est donc crucial pour obtenir des formules CNF de taille raisonnable. Nous proposons une nouvelle borne supérieure basée sur une propriété syntaxique de la formule que nous appelons *diamond degree*. Nous fournissons quelques preuves expérimentales que notre approche améliore significativement les solveurs S5 de la littérature.

Le reste de l'article est organisé comme suit : nous présentons dans un premier temps la logique modale S5 ainsi que les différentes borne sur la taille d'un modèle S5 ; nous détaillons ensuite la réduction de S5-SAT à SAT, paramétrée par le nombre de mondes à considérer. Nous présentons dans un deuxième temps deux améliorations pour réduire la taille de l'encodage SAT : une meilleure borne supérieure sur le nombre de mondes à considérer ainsi qu'un *caching* structurel. Enfin, nous comparons expérimentalement l'efficacité de notre approche aux solveurs S5 existants.

Préliminaires

Soit $\mathbb{P}$ un ensemble fini non vide de variables propositionnelles. Le langage $\mathcal{L}$ de la logique modale S5 est l'ensemble des formules ϕ définies par la grammaire suivante :

$$\phi ::= \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \Box\phi \mid \Diamond\phi$$

où p est une variable de $\mathbb{P}$. Une formule de la forme $\Box\phi$ (*box phi*) signifie que ϕ est nécessairement vrai. Une formule de la forme $\Diamond\phi$ (*diamond phi*) signifie que ϕ est possiblement vrai. Toute formule $\phi \in \mathcal{L}$ peut être convertie en une formule équivalente en forme normale négative (NNF), c'est-à-dire que les négations apparaissent seulement devant les variables propositionnelles (voir la définition dans [32]), notée $\text{nnf}(\phi)$. Cela peut se faire en temps polynomial. Les formules de $\mathcal{L}$ sont interprétées en utilisant des modèles S5 pointés. Une modèle S5 est une paire (W, V), où W est un ensemble non vide de mondes possibles et V est une fonction de valuation de W dans $(\mathbb{P} \to \{0, 1\})$. Un modèle S5 pointé est un triplet (W, V, w), où (W, V) est un modèle S5 et $w \in W$. La relation de satisfaction $\models$ entre des formules du langage $\mathcal{L}$ et des modèles S5 pointés est définie récursivement comme suit :

$(W, V, w) \models \top$
$(W, V, w) \models p$ ssi $V(w)(p) = 1$
$(W, V, w) \models \neg\phi$ ssi $(W, V, w) \not\models \phi$
$(W, V, w) \models \phi_1 \wedge \phi_2$ ssi $(W, V, w) \models \phi_1$ et $(W, V, w) \models \phi_2$
$(W, V, w) \models \phi_1 \vee \phi_2$ ssi $(W, V, w) \models \phi_1$ ou $(W, V, w) \models \phi_2$
$(W, V, w) \models \Box\phi$ ssi pour tout $w' \in W$, nous avons $(W, V, w') \models \phi$
$(W, V, w) \models \Diamond\phi$ ssi il existe un $w' \in W$ tel que $(W, V, w') \models \phi$

La validité et la satisfiabilité sont définies comme d'habitude. Une formule $\phi \in \mathcal{L}$ est valide, noté $\models \phi$, si et seulement si, pour tout modèle S5 pointé (W, V, w), nous avons $(W, V, w) \models \phi$. De plus, ϕ est satisfiable si et seulement si $\not\models \neg\phi$.

Exemple 1 *Voici un exemple d'une formule en logique modale S5* $\phi = (\Diamond(p_1 \wedge \Box p_2) \wedge \Diamond p_3)$ *ainsi qu'un exemple de modèle* (W, V, w) *de tel sorte que* $(W, V, w) \models \phi$

$$W = \{w, v, u\}$$
$$V(w) = \{(p_1, 1), (p_2, 1), (p_3, 0)\}$$
$$V(u) = \{(p_1, 0), (p_2, 1), (p_3, 0)\}$$
$$V(v) = \{(p_1, 0), (p_2, 1), (p_3, 1)\}$$

De S5-SAT à SAT

Il a été montré dans [24] que si une formule S5 ϕ contenant n connecteurs modaux est satisfiable, alors il existe un modèle S5 satisfaisant ϕ contenant au plus $n + 1$ mondes. Nous savons aussi qu'il existe un algorithme s'exécutant en temps polynomial capable de transformer un problème S5-SAT en un problème SAT (vu que S5-SAT est NP-complet [24]). Cependant, à notre connaissance, personne n'a évalué cette approche dans la pratique. Celle-ci n'a pas été comparée aux solveurs de la littérature jusqu'à présent. Pourtant, les approches basées sur SAT sont connues pour être très efficaces en pratique.

Un codage SAT est ici représenté par une fonction de traduction tr, qui prend en entrée une formule S5 ϕ ainsi qu'un nombre de mondes n dans un modèle S5 et qui produit une formule propositionnelle. Ceci est inspiré par la

traduction standard vers la logique du premier ordre [31]. [1]

$$\mathrm{tr}(\phi, n) = \mathrm{tr}'(\mathrm{nnf}(\phi), 1, n)$$
$$\mathrm{tr}'(\top, i, n) = \top$$
$$\mathrm{tr}'(\neg\top, i, n) = \neg\top$$
$$\mathrm{tr}'(p, i, n) = p_i$$
$$\mathrm{tr}'(\neg p, i, n) = \neg p_i$$
$$\mathrm{tr}'((\phi \wedge \cdots \wedge \delta), i, n) = \mathrm{tr}'(\phi, i, n) \wedge \cdots \wedge \mathrm{tr}'(\delta, i, n)$$
$$\mathrm{tr}'((\phi \vee \cdots \vee \delta), i, n) = \mathrm{tr}'(\phi, i, n) \vee \cdots \vee \mathrm{tr}'(\delta, i, n)$$
$$\mathrm{tr}'(\Box\phi, i, n) = \bigwedge_{j=1}^{n} (\mathrm{tr}'(\phi, j, n))$$
$$\mathrm{tr}'(\Diamond\phi, i, n) = \bigvee_{j=1}^{n} (\mathrm{tr}'(\phi, j, n))$$

La traduction ajoute de nouvelles variables booléennes p_i à la formule, désignant la valeur de vérité de p dans le monde w_i. Dans cette fonction, le paramètre i représente l'indice du monde. La fonction est définie sur une formule en forme normale négative (NNF) par souci de simplicité.

Exemple 2 *Soit la formule $\phi = \Diamond(a \wedge \Box b)$, la Figure 1 montre la traduction de la formule ϕ avec $n = 2$.*

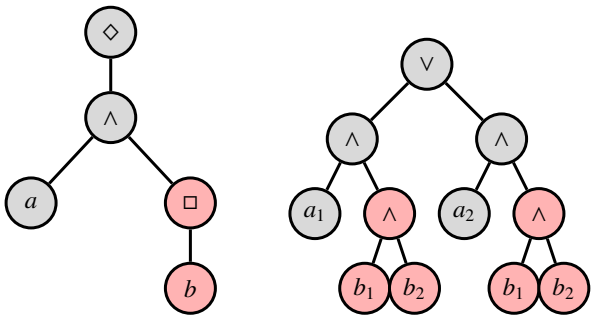


FIGURE 1 – De S5 ϕ (gauche) vers la logique propositionnelle avec n=2 (droite)

Si la valeur de n est une borne supérieure du nombre de mondes dans le modèle, alors la traduction et la formule S5 d'origine sont équi-satisfiables. C'est en particulier le cas pour la borne supérieure montrée dans [24] :

Définition 1 *Soit $\phi \in \mathcal{L}$, $\mathtt{nm}(\phi)$ désigne le nombre de connecteurs modaux contenus dans le formule ϕ.*

Théorème 1 *Une formule ϕ de $\mathcal{L}$ est satisfiable si et seulement si $tr(\phi, \mathtt{nm}(\phi) + 1)$ est satisfiable.*

Démonstration 1 *Le théorème 1 est prouvé de la même façon que la traduction standard vers la logique du premier ordre, plus le Lemme 6.1 de [24].*

1. Notons que la relation d'accessibilité n'a pas besoin d'être représentée dans un modèle S5, puisque celle-ci est une relation d'équivalence.

Notons que le résultat de la traduction n'est pas en CNF. Ainsi, la traduction classique en CNF utilisant l'algorithme de Tseitin [37] est nécessaire pour utiliser un oracle SAT.

Amélioration de la borne supérieure

La taille du codage dépend de $\mathtt{nm}(\phi)$. En pratique, une telle borne supérieure produit des formules déraisonnablement grandes. Une première étape consiste donc à améliorer cette borne supérieure. Nous proposons d'utiliser une nouvelle mesure appelée *diamond degree* comme nouvelle borne supérieure.

Définition 2 (*Diamond degree*) *Le diamond degree d'une formule ϕ de $\mathcal{L}$, noté $\mathtt{dd}(\phi)$, est défini récursivement comme suit :*

$$\mathtt{dd}(\phi) = \mathtt{dd}'(\mathrm{nnf}(\phi))$$
$$\mathtt{dd}'(\top) = 0$$
$$\mathtt{dd}'(\neg\top) = 0$$
$$\mathtt{dd}'(p) = 0$$
$$\mathtt{dd}'(\neg p) = 0$$
$$\mathtt{dd}'(\phi \wedge \psi) = \mathtt{dd}'(\phi) + \mathtt{dd}'(\psi)$$
$$\mathtt{dd}'(\phi \vee \psi) = \max(\mathtt{dd}'(\phi), \mathtt{dd}'(\psi))$$
$$\mathtt{dd}'(\Box\phi) = \mathtt{dd}'(\phi)$$
$$\mathtt{dd}'(\Diamond\phi) = 1 + \mathtt{dd}'(\phi)$$

Le calcul du *diamond degree* nécessite la conversion de ϕ en NNF. Comme mentionné dans la section précédente, cette opération peut être effectuée en temps et espace polynomial. Ainsi, le *diamond degree* de toute formule peut être calculé en temps polynomial. Sans perte de généralité, nous supposerons que ϕ est en NNF et considérerons $\mathtt{dd}'$ au lieu de $\mathtt{dd}$.

De façon informelle, le *diamond degree* représente une borne supérieure sur le nombre de diamants à prendre en compte pour satisfaire la formule. Pour montrer que le *diamond degree* est une borne supérieure valide pour notre codage SAT, nous utiliserons une méthode des tableaux. Nous avons donc besoin de définitions additionnelles. Soient ϕ une formule en NNF et $\mathrm{sub}(\phi)$ l'ensemble des sous-formules de ϕ. Un tableau pour ϕ est un ensemble non vide $T = \{s_0, s_1, \ldots, s_n\}$ tel que $s_i \in T$ est un sous-ensemble de $\mathrm{sub}(\phi)$ et $\phi \in s_0$. De plus, tout ensemble $s \in T$ satisfait les conditions suivantes :

1. $\neg\top \notin s$.
2. si $p \in s$ alors $\neg p \notin s$.
3. si $\neg p \in s$ alors $p \notin s$.
4. si $\psi_1 \wedge \psi_2 \in s$ alors $\psi_1 \in s$ et $\psi_2 \in s$.
5. si $\psi_1 \vee \psi_2 \in s$ alors $\psi_1 \in s$ ou $\psi_2 \in s$.
6. si $\Box\psi_1 \in s$ alors $\forall s' \in T$ nous avons $\psi_1 \in s'$.

7. si $\Diamond\psi_1 \in s$ alors $\exists s' \in T$ tel que $\psi_1 \in s'$.

Lemme 1 *Soit ϕ une formule en NNF. ϕ est satisfiable si et seulement si il existe un tableau pour ϕ.*

Démonstration 2 *Nous pouvons démontrer ce résultat de manière standard. De la gauche vers la droite, nous montrons que chaque règle du tableau préserve sa satisfiabilité. De la droite vers la gauche, nous montrons par induction sur la structure de ϕ que si le tableau pour ϕ existe, alors nous pouvons construire un modèle pour ϕ en utilisant chaque s comme monde possible de ce modèle.*

Lemme 2 *Soit ϕ une formule en NNF. Le nombre d'éléments de l'ensemble T créés en construisant le tableau de ϕ est borné par $\mathsf{dd}'(\phi) + 1$.*

Démonstration 3 *Soit une formule $\psi \in \text{sub}(\phi)$. Soit $g(\psi)$ le nombre d'ensembles s ajouté à T par la présence de ψ. Autrement dit, $g(\psi)$ est le nombre de fois où la condition impliquant l'opérateur $\Diamond$ est déclenchée pour une sous-formule de ψ. Nous montrons que, pour toute formule $\psi \in \text{sub}(\phi)$, nous avons $g(\psi) \leq \mathsf{dd}'(\psi)$.*

Pour ce faire, nous utilisons une induction sur la structure de ψ.

Base d'induction. Nous considérons quatre cas : (1) $\psi = \top$, (2) $\psi = \neg\top$, (3) $\psi = p$ et (4) $\psi = \neg p$. Dans chaque cas, le condition impliquant $\Diamond$ ne sera jamais déclenché par des formules de sub(ψ). *Ainsi, $g(\psi) = 0 \leq \mathsf{dd}'(\psi)$.*

Étape d'induction. Nous considérons quatre cas :

1. *$\psi = \psi_1 \wedge \psi_2$. Supposons que $\psi \in s$, pour un certain $s \in T$. Dans ce cas, l'algorithme ajoute ψ_1 et ψ_2 à s. Par conséquent, $g(\psi)$ est borné par $g(\psi_1) + g(\psi_2)$. Ainsi, $g(\psi)$ est borné par $\mathsf{dd}'(\psi_1) + \mathsf{dd}'(\psi_2)$ (d'après l'hypothèse d'induction). D'où $g(\psi) \leq \mathsf{dd}'(\psi)$.*
2. *$\psi = \psi_1 \vee \psi_2$. Supposons que $\psi \in s$, pour un certain $s \in T$. Dans ce cas, l'algorithme ajoute soit ψ_1 soit ψ_2 à s. Par conséquent, $g(\phi)$ est borné par $\max(g(\psi_1), g(\psi_2))$. Ce dernier est borné par $\max(\mathsf{dd}'(\psi_1), \mathsf{dd}'(\psi_2))$ (d'après l'hypothèse d'induction). Ainsi $g(\psi) \leq \mathsf{dd}'(\psi)$.*
3. *$\psi = \Box\psi_1$. Supposons que $\psi \in s$, pour un certain $s \in T$. Dans ce cas, l'algorithme ajoute ψ_1 à tout $s' \in T$. $g(\psi)$ est donc borné par $g(\psi_1)$, qui est borné par $\mathsf{dd}'(\psi_1)$ (d'après l'hypothèse d'induction). Ainsi $g(\psi) \leq \mathsf{dd}'(\psi)$.*
4. *$\psi = \Diamond\psi_1$. Supposons que $\psi \in s$, pour un certain $s \in T$. Dans ce cas, s'il n'existe aucun s' contenant ψ_1, alors l'algorithme ajoute un nouvel ensemble s'', contenant ψ_1, à T. Par conséquent, $g(\psi)$ est borné par $1 + g(\psi_1)$. Ce dernier est borné par $1 + \mathsf{dd}'(\psi_1)$ (d'après l'hypothèse d'induction). Ainsi $g(\psi) \leq \mathsf{dd}'(\psi)$.*

Nous avons donc $|T| = 1 + g(\phi) \leq 1 + \mathsf{dd}'(\phi)$.

Ainsi, pour toute formule $\phi \in \mathcal{L}$, chaque ensemble s_i du tableau T correspond à un monde $w_i \in W$ dans le modèle S5. $|T| \leq \mathsf{dd}(\phi) + 1$ signifie que le nombre de mondes dans le modèle S5 est borné par $\mathsf{dd}(\phi) + 1$.

Théorème 2 *Une formule $\phi \in \mathcal{L}$ est satisfiable si et seulement si $tr(\phi, \mathsf{dd}(\phi) + 1)$ est satisfiable.*

Caching structurel

Le *caching* est un moyen classique d'éviter du travail redondant. *SAT effectue le *caching* en utilisant une « matrice de bits » [16]. L'implémentation efficace des bibliothèques de BDD [6] repose également sur le *caching*, pour construire un graphe explicite. Ces deux exemples nécessitent plus de temps et d'espace pour la recherche et la mise en cache de travaux exécutés ultérieurement. Ici, notre technique est un compromis « simple mais efficace ». Il ne mémorise pas le travail, de sorte qu'il ne peut pas mettre en cache toutes les formules possibles, mais il ne nécessite qu'un indicateur pour détecter le travail redondant. En d'autres termes, nous ne générons pas la formule redondante, mais grâce à un indicateur, nous sommes capable de détecter les sous-formules redondantes.

Dans l'exemple représenté sur figure 2.b, la sous-formule $(b_1 \wedge b_2)$ apparaît deux fois. La traduction du premier diamant crée deux sous-formules $(a_1 \wedge \Diamond b)$ et $(a_2 \wedge \Diamond b)$, où chaque $\Diamond b$ doit être traduit.

Puisque nous sommes en S5 (tous les mondes sont connectés), les traductions de $\Diamond b$ sur des mondes différents sont équivalentes, nous pouvons donc réutiliser la même sous-formule. Cela signifie qu'au lieu d'utiliser un arbre nous pouvons travailler avec un DAG, ce qui permet une traduction en CNF plus efficace.

Lemme 3 $\text{tr}'(\circ\phi, i, n) = \text{tr}'(\circ\phi, j, n)\ \forall i, j$ *et* $\circ \in \{\Diamond, \Box\}$

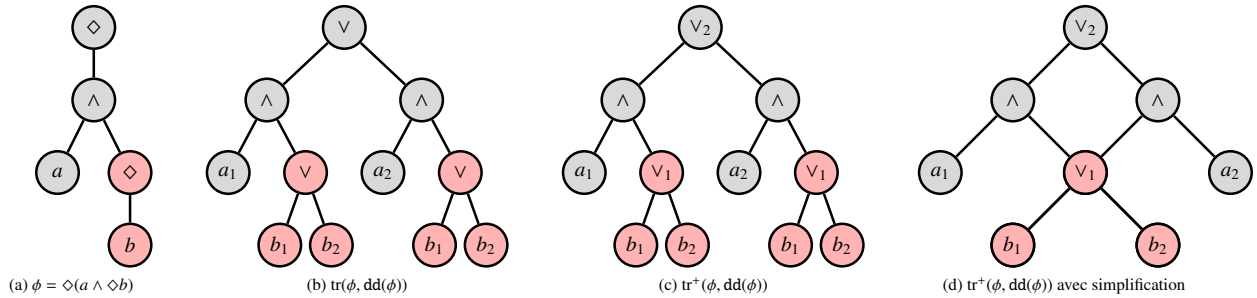
Démonstration 4 *nous avons deux cas à considérer :*

— *Si ($\circ = \Box$), alors* $\text{tr}'(\Box\phi, i, n) = \bigwedge_{k=1}^{n}(\text{tr}'(\phi, k, n))$

— *Si ($\circ = \Diamond$), alors* $\text{tr}'(\Diamond\phi, i, n) = \bigvee_{k=1}^{n}(\text{tr}'(\phi, k, n))$

Dans chaque cas, le résultat est indépendant de i, ainsi choisir j comme indice donne le même résultat.

De façon informelle, le Lemme 3 montre le fait que, quelle que soit la façon dont est incorporée la sous-formule modale, sa traduction donnera toujours le même résultat (indépendamment de l'indice *i*). Par conséquent, nous pouvons commencer par traduire la formule la plus profonde, marquer le nœud correspondant puis revenir en arrière. La formule résultante peut contenir plusieurs nœuds ayant le même marquage. Cela signifie que ces sous-formules sont syntaxiquement identiques (voir Figure 2.c). Ensuite, nous ne conservons qu'une occurence de la sous-formule, tranformant l'arbre en un DAG (voir Figure 2.d). Le *caching*



(a) $\phi = \Diamond(a \wedge \Diamond b)$ (b) $\mathrm{tr}(\phi, \mathrm{dd}(\phi))$ (c) $\mathrm{tr}^+(\phi, \mathrm{dd}(\phi))$ (d) $\mathrm{tr}^+(\phi, \mathrm{dd}(\phi))$ avec simplification

FIGURE 2 – Traduction de $\Diamond(a \wedge \Diamond b)$ (a), traduction initiale (b), formule marquée (c), traduction finale (d)

structurel est donc effectuée à la volée avant de traduire en CNF. La fonction de traduction utilisant cette technique est notée tr^+.

Expérimentations

Nous avons considéré LCKS5TabProver [1] et SPASS 3.7 [39] qui sont, à notre connaissance, à la pointe de la résolution du problème de satisfiabilité en logique modale S5. Nous les avons comparés à quatre configurations différentes de S52SAT : `nm`, `nm+` (avec *caching*), `dd`, `dd+` (avec *caching*) (`http://www.cril.univ-artois.fr/~montmirail/s52SAT`). Afin d'évaluer rigoureusement les solveurs, nous avons utilisé la même entrée pour chacun d'eux, au format InToHyLo [22]. Pour ce faire, nous avons modifié le code de LCKS5TabProver pour le rendre capable de lire ce format. Les modifications sont assez simples pour garantir que les performances du solveur ne sont pas affectées.

Nous avons utilisé SPASS 3.7 au lieu de la dernière version (3.9) car le premier lit le format `dfg`, qui peut être produit à partir de benchmarks InToHyLo en utilisant l'outil `ftt` (Fast Transformation Tool) intégré dans Spartacus [19]. La différence entre 3.7 et 3.9 est minimale selon le site web du solveur. Le temps de traduction est négligeable dans nos expérimentations. Nous avons également considéré MetTel2 [36] et LoTREC [12] pour les solveurs de l'état de l'art en logique modale S5, mais ils ne sont malheureusement pas conçus pour résoudre efficacement les problèmes de logique S5. Nous utilisons Glucose 4.0 [3] comme solveur SAT back-end.

Nous avons évalué ces solveurs sur des benchmarks bien établis de la logique modale : $\mathrm{3CNF}_K$ [30], MQBF_K [26], $\mathrm{TANCS2000}_K$ [27] et $\mathrm{LWB}_{K,KT,S4}$ [4].

Notons qu'ils sont conçus pour les logiques modales K, KT et S4. Par conséquent, certains d'entre eux sont triviaux dans le cadre S5. Toutefois, nous pensons que les résultats obtenus sur ces benchmarks demeurent importants. S5-SAT implique K, KT et S4-SAT, ainsi nous pourrions envisager de résoudre la satisfiabilité en S5 comme étape de prétraitement pour ces logiques modales.

Nous avons fixé la limite de mémoire à 8Go et la limite d'exécution à 900 secondes. Nous avons rarement atteint ce délai (804 fois sur 12444 essais). La plupart des benchmarks non résolus sont dus au manque de mémoire.

Dans les tables suivantes, nous fournissons le nombre de benchmarks résolus, en gras les meilleurs résultats d'une rangée/colonne donnée (selon l'orientation du tableau) ; et entre parenthèses, nous fournissons le nombre de benchmarks qui ne peuvent être résolus à cause du manque de mémoire.

3CNF_K : le *caching* n'aide pas

Solveur	d=2	d=4	d=6	Total
LckS5TabProver	0 (17)	0 (29)	0 (40)	0
S52SAT `nm`	21 (24)	0 (45)	0 (45)	21
S52SAT `nm+`	21 (24)	0 (45)	0 (45)	21
S52SAT `dd`	**45 (0)**	**8 (27)**	0 (45)	**53**
S52SAT `dd+`	**45 (0)**	**8 (27)**	0 (45)	**53**
SPASS 3.7	**45 (0)**	5 (40)	0 (45)	50

TABLE 1 – Nombre d'instances résolues en $\mathrm{3CNF}_K$

Les résultats sont affichés dans la table 1. `dd`, `dd+` et SPASS sont assez proches les uns des autres. Les formules se composent de grandes conjonctions où chaque conjoint a une profondeur modale d'au plus 2, 4 ou 6. Elles sont construites de telle manière que notre algorithme de *caching* ne trouve pas de redondances sur ces formules. C'est pourquoi le *caching* ne fournit aucun avantage sur ces benchmarks.

modKSSS et modKLadn : le *caching* aide

n représente le nombre de variables et a représente le nombre d'alternances dans le préfixe QBF d'origine, voir [26] pour plus de détails, et # correspond au nombre de benchmarks disponible pour une paire (n,a) donnée.

n,*a*	#	LckS5...	nm	nm+	dd	dd+	SPASS 3.7
4,4	40	0 (12)	32	**40**	**40**	**40**	**40**
4,6	40	0 (23)	32	**40**	**40**	**40**	32 (8)
8,4	40	0 (16)	32	**40**	39 (1)	**40**	16 (24)
8,6	40	0 (17)	24	**40**	**40**	**40**	10 (30)
16,4	40	0 (10)	22	**40**	**40**	**40**	8 (32)
16,6	40	0 (16)	19	**40**	39 (1)	**40**	2 (38)
total	240	0	161	**240**	238	**240**	108
4,4	40	**40**	0 (40)	**40**	0 (40)	**40**	0 (40)
4,6	40	14 (0)	0 (40)	32 (8)	0 (40)	**40**	0 (40)
8,4	40	2 (0)	0 (40)	8 (32)	0 (40)	**40**	0 (40)
8,6	40	0 (0)	0 (40)	0 (40)	0 (40)	**8** (32)	0 (40)
16,4	40	0 (0)	0 (40)	0 (40)	0 (40)	0 (40)	0 (40)
16,6	40	0 (0)	0 (40)	0 (40)	0 (40)	0 (40)	0 (40)
total	240	56	0	80	0	**128**	0

TABLE 2 – Au-dessus : modKSSS | En-dessous : modKLadn

Dans cette catégorie, nous pouvons voir que dd et dd+ sont beaucoup plus efficaces que SPASS. Les formules de ces benchmarks contiennent beaucoup de redondances : nous pouvons voir que l'utilisation du *caching* nous permet de résoudre tous les benchmarks modKSSS par exemple.

TANCS-2000 : Coûteux en mémoire

n,*a*	#	LckS5...	nm	nm+	dd	dd+	SPASS 3.7
4,4	40	0 (38)	**40**	**40**	**40**	**40**	**40**
4,6	40	0 (33)	**40**	**40**	**40**	**40**	**40**
8,4	40	0 (38)	**40**	**40**	**40**	**40**	**40**
8,6	40	0 (39)	**40**	**40**	**40**	**40**	**40**
16,4	40	0 (40)	**40**	**40**	**40**	**40**	**40**
16,6	40	0 (40)	**40**	**40**	**40**	**40**	35 (5)
total	240	0	**240**	**240**	**240**	**240**	235
4,4	40	23 (0)	3 (37)	**40**	**40**	**40**	**40**
4,6	40	3 (0)	0 (40)	**40**	**40**	**40**	31 (9)
total	80	26	3	**80**	**80**	**80**	71

TABLE 3 – TANCS-2000. Au-dessus : qbfMS | En-dessous : qbfML

Ici, nous pouvons voir que ces problèmes peuvent être résolus par presque tous les solveurs. Les instances non résolues par SPASS 3.7 le sont à cause du manque de mémoire. Comme indiqué dans la section suivante, en augmentant la limite mémoire à 32Go, ces instances sont résolues par SPASS.

LWB K, KT, S4 : dd et le *caching* aident

Les benchmarks sont à l'origine séparés sur les formules SAT/UNSAT pour les logiques K, KT et S4. Évidemment, la satisfaction dans S5 peut différer, nous avons donc supprimé la séparation SAT/UNSAT. Les résultats sont affichés dans la table 4. Là encore, dd+ est légèrement meilleur que SPASS. Les benchmarks qui ne peuvent être résolus sont un encodage spécifique en logique modale du principe de *pigeon hole* [20] dans la logique correspondante.

Solveur	Logique K	Logique KT	Logique S4	Total
LckS5TabProver	227 (73)	206 (102)	194 (111)	627
S52SAT nm	307 (66)	344 (33)	292 (86)	943
S52SAT nm+	351 (21)	363 (12)	349 (28)	1063
S52SAT dd	333 (40)	355 (23)	337 (40)	1025
S52SAT dd+	**357 (12)**	**364 (12)**	**363 (12)**	**1084**
SPASS 3.7	343 (16)	363 (15)	360 (17)	1066

TABLE 4 – Nombre d'instances résolues dans $LWB_{K,KT,S4}$

Résultats globaux sur tous les benchmarks

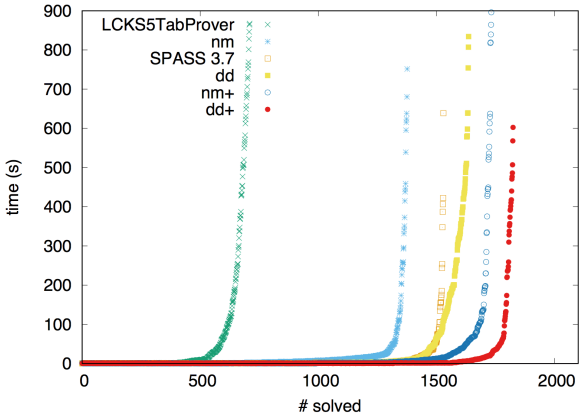


FIGURE 3 – Distribution des temps d'exécution

Solveur	# résolus	# SAT	MO	TO
LckS5TabProver	709	143	710	655
S52SAT nm	1377	411	667	30
S52SAT nm+	1733	452	292	49
S52SAT dd	1645	433	412	17
S52SAT dd+	**1834**	**460**	**203**	37
SPASS 3.7	1530	451	528	**16**

TABLE 5 – Résultats globaux sur tous les benchmarks

SPASS est moins rapide, en partie en raison de ses mauvais résultats sur les benchmarks modKSSS et modKLadn. Bien que l'encodage SAT par défaut épuise souvent la mémoire disponible, chacune des deux améliorations proposées réduit considérablement le nombre de *memory-out*, les meilleurs résultats étant obtenus lorsque les deux sont activés.

Il semble assez clair que le *caching* est essentiel dans notre approche pour résoudre efficacement ces benchmarks. Ceci est attesté par le diagramme de dispersion de la figure 4. L'axe des abscisses correspond au temps utilisé par dd et l'axe des ordonnées correspond au temps utilisé par dd+ pour résoudre le problème.

La principale raison de l'amélioration est la réduction de la taille du codage CNF, comme le montre la table 6. Nous

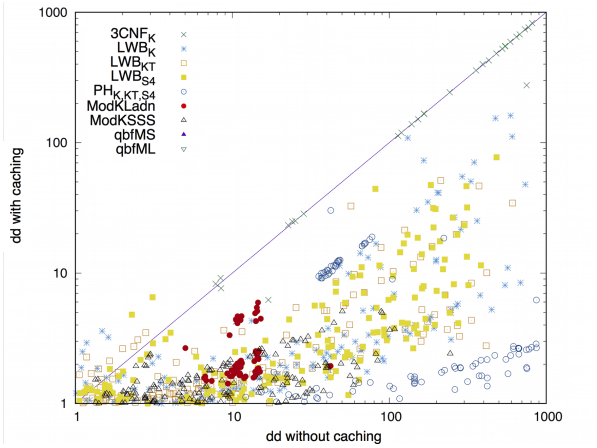


FIGURE 4 – Temps d'exécution de dd avec et sans *caching*

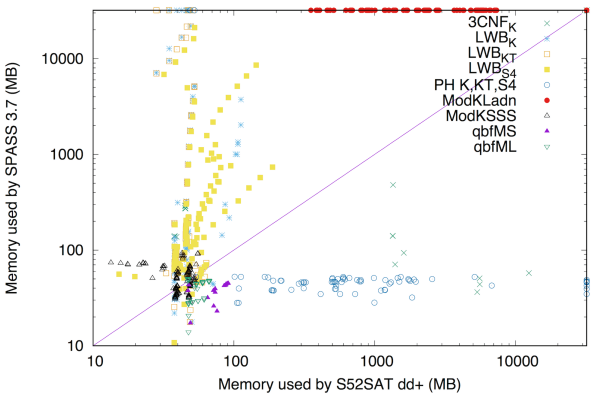


FIGURE 5 – Comparaison (32 Go) SPASS 3.7 vs dd+

Solveur	moy	médiane	max
nm	6 881 821	1 100 054	58 653 264
nm+	1 619 923	118 040	29 492 779
dd	2 385 515	169 324	55 813 557
dd+	269 891	27 090	22 914 442

TABLE 6 – Nombre de clauses dans les formules CNF générées

avions une valeur médiane de 1 100 054 clauses pour nm, et cette valeur tombe à 27 090 pour dd+ sur exactement les mêmes problèmes.

Importance de la mémoire

Nous avons répété les expérimentations avec 32Go de RAM. Notons que pour l'approche SAT, le manque de mémoire se produit pendant la phase de traduction, tandis que pour les autres, la mémoire est épuisée dans la phase de résolution.

Nous nous demandions ce que serait la performance avec plus de mémoire. La table 7 résume le nombre de problèmes résolus avec 8Go et 32Go par chaque solveur. Fournir 32Go à SPASS ne change pas les résultats de manière significative : il a résolu 30 instances supplémentaires. Notre approche SAT tire avantage de cette quantité de mémoire accrue, jusqu'à 98 benchmarks supplémentaires peuvent être résolus par nm sans *caching*. 32Go sont suffisants pour LckS5TabProver (sans *memory-out*), mais seulement une instance supplémentaire peut être résolue.

Solveur	8Go	32Go	MO avec 32Go
LckS5TabProver	709	710	0
SPASS 3.7	1530	1560	498
S52SAT nm	1377	1475	382
S52SAT nm+	1733	1800	166
S52SAT dd	1645	1672	317
S52SAT dd+	1834	1888	96

TABLE 7 – Nombre d'instances résolues avec 8Go et 32Go

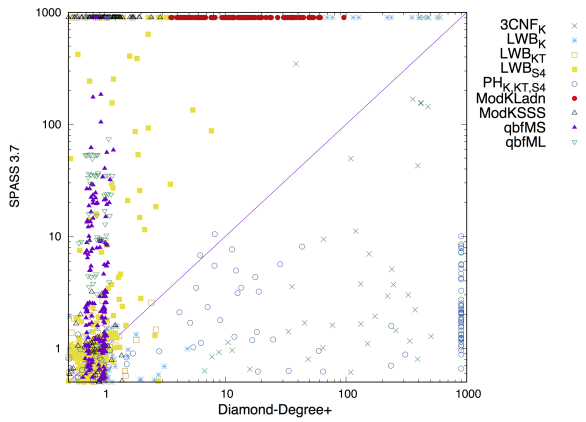


FIGURE 6 – Temps d'exécution de SPASS 3.7 vs. dd+

La figure 5 compare la consommation de mémoire de SPASS 3.7 et S52SAT dd+ en Go. SPASS nécessite généralement plus de mémoire que dd+. Le temps d'exécution et la consommation de mémoire sont légèrement en corrélation (le coefficient de corrélation de Pearson pour dd+ est égal à 0.58 et celui de SPASS est de 0.57). Dans la catégorie modKLadn, SPASS manque de mémoire, même avec 32Go.

Une perspective intéressante ici serait d'établir une heuristique qui détermine, avant même de réaliser la traduction, que la génération de la CNF va dépasser la limite de mémoire autorisée. Une telle heuristique pourrait être '$(\text{len}(\phi) \times \text{dd}(\phi)) > \mathit{BIGNUMBER}$'. On pourrait ensuite

faire le choix par exemple, de ne prendre que $\frac{\mathtt{dd}(\phi)}{2}+1$ comme borne supérieure, si nous trouvons une solution, c'est une solution de la formule complète, sinon il faudrait augmenter au fur et à mesure le nombre de monde autorisée jusqu'à arriver soit à une solution, soit au MO qui aurait du se produire sans l'heuristique. Une telle approche se rapprocherait très fortement des approches CounterExample Guided Abstration Refinement (CEGAR) [8].

Comparaison avec la littérature

La figure 6 montre une comparaison détaillée du temps d'exécution entre dd+ et SPASS sur tous les benchmarks. Dans la plupart des cas, notre approche surpasse SPASS. Les deux cas où elle est moins efficace sont des problèmes *pigeon hole* UNSAT combinatoires durs et des benchmarks dits « 3CNF » pour lesquels ni le *diamond degree* ni le *caching* ne sont utiles.

Conclusion

Nous avons présenté un nouvel encodage SAT pour résoudre le problème S5-SAT en utilisant un solveur SAT. Il est basé sur une réduction de S5-SAT à SAT avec deux améliorations : une meilleure borne supérieure sur le nombre de mondes requis et un *caching* structurel. Nous avons comparé notre approche aux solveurs représentant, à notre connaissance, l'état de l'art pour la résolution pratique de S5-SAT, sur un large éventail de benchmarks. L'approche basée sur SAT avec toutes les améliorations a permis de surclasser tous ces solveurs.

Même si les benchmarks peuvent ne pas être représentatifs de l'utilisation de S5 en pratiques puisqu'ils proviennent d'autres logiques modales (K, KT, S4), ces résultats ouvrent des perspectives intéressantes. En effet, prouver la satisfiabilité d'une formule de la logique modale S5 implique que la formule est également satisfiable dans des systèmes moins restrictifs (c'est-à-dire dans K, KT, S4). Puisque notre solveur S5 fournit un modèle S5 en quelques secondes (temps médian de 2,06s), nous pourrions parfaitement l'utiliser comme une étape de prétraitement pour résoudre les benchmarks dans d'autres logiques modales.

Les résultats préliminaires dans cette direction sont assez encourageants : sur 276 benchmarks satisfiables en logique S5 (donc en K), S52SAT surpasse Spartacus [19] sur 158 d'entre eux. Une autre perspective serait d'adapter S52SAT afin de résoudre les problèmes KD45-SAT étant donné que KD45 est également NP-complet [9].

Remerciements

Nous remercions Renate Schmidt pour son aide dans le recherche de solveurs S5. Une partie de ces travaux ont étés soutenues par le Ministère de l'Enseignement Supérieur et de la Recherche et le Conseil Régional Nord-Pas de Calais par le biais du Contrat de Plan d'Epargne (CPER) ainsi que par une subvention EC FEDER.

Informations

Ces travaux ont déjà étés publiés à la conférence AAAI'17 [7].

Références

[1] Pietro Abate, Rajeev Goré, and Florian Widmann. Cut-free single-pass tableaux for the logic of common knowledge. In *Workshop on Agents and Deduction at TABLEAUX'07*, 2007.

[2] Carlos Areces, Pascal Fontaine, and Stephan Merz. Modal Satisfiability via SMT Solving. In *Software, Services, and Systems - Essays Dedicated to Martin Wirsing*, pages 30–45, 2015.

[3] Gilles Audemard and Laurent Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *Proc. of IJCAI'09*, pages 399–404, 2009.

[4] Peter Balsiger, Alain Heuerding, and Stefan Schwendimann. A Benchmark Method for the Propositional Modal Logics K, KT, S4. *J. Autom. Reasoning*, 24(3) :297–317, 2000.

[5] Meghyn Bienvenu, Hélène Fargier, and Pierre Marquis. Knowledge Compilation in the Modal Logic S5. In *Proc. of AAAI'10*, 2010.

[6] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8) :677–691, 1986.

[7] Thomas Caridroit, Jean-Marie Lagniez, Daniel Le Berre, Tiago de Lima, and Valentin Montmirail. A SAT-based Approach For Solving The Modal Logic S5-Satisfiability Problem. In *Proc. of AAAI'17*, 2017.

[8] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5) :752–794, 2003.

[9] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. The MIT Press, 1995.

[10] Matt Fairtlough and Michael Mendler. An Intuitionistic Modal Logic with Applications to the Formal Verification of Hardware. In *Proc. of CSL'94*, pages 354–368, 1994.

[11] Melvin Fitting. Modality and Databases. In *Proc. of TABLEAUX'00*, pages 19–39, 2000.

[12] Olivier Gasquet, Andreas Herzig, Dominique Longin, and Mohamad Sahade. LoTREC : Logical Tableaux

Research Engineering Companion. In *Proc. of TABLEAUX'05*, pages 318–322, 2005.

[13] Enrico Giunchiglia, Fausto Giunchiglia, and Armando Tacchella. The SAT-Based Approach for Classical Modal Logics. In *Proc. of AI*IA'99*, pages 95–106, 1999.

[14] Enrico Giunchiglia, Roberto Sebastiani, Fausto Giunchiglia, and Armando Tacchella. SAT vs. Translation based decision procedures for modal logics : a comparative evaluation. *Journal of Applied Non-Classical Logics*, 10(2) :145–172, 2000.

[15] Enrico Giunchiglia and Armando Tacchella. System description : *SAT : A platform for the development of modal decision procedures. In *Proc. of CADE'00*, pages 291–296. Springer, 2000.

[16] Enrico Giunchiglia and Armando Tacchella. A Subset-Matching Size-Bounded Cache for Testing Satisfiability in Modal Logics. *Ann. Math. Artif. Intell.*, 33(1) :39–67, 2001.

[17] Enrico Giunchiglia, Armando Tacchella, and Fausto Giunchiglia. SAT-Based Decision Procedures for Classical Modal Logics. *J. Autom. Reasoning*, 28(2) :143–171, 2002.

[18] Fausto Giunchiglia and Roberto Sebastiani. Building Decision Procedures for Modal Logics from Propositional Decision Procedures : The Case Study of Modal K(m). *Inf. Comput.*, 162(1-2) :158–178, 2000.

[19] Daniel Götzmann, Mark Kaminski, and Gert Smolka. Spartacus : A Tableau Prover for Hybrid Logic. *Electr. Notes Theor. Comput. Sci.*, 262 :127–139, 2010.

[20] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(0) :297 – 308, 1985.

[21] Joseph Y. Halpern and Leandro Chaves Rêgo. Characterizing the NP-PSPACE Gap in the Satisfiability Problem for Modal Logic. In *Proc. of IJCAI'07*, pages 2306–2311, 2007.

[22] Guillaume Hoffmann. *Tâches de raisonnement en logiques hybrides*. PhD thesis, Henri Poincaré University, Nancy, 2010.

[23] Mark Kaminski and Tobias Tebbi. InKreSAT : Modal Reasoning via Incremental Reduction to SAT. In *Proc. of CADE'13*, pages 436–442, 2013.

[24] Richard E. Ladner. The Computational Complexity of Provability in Systems of Modal Propositional Logic. *SIAM J. Comput.*, 6(3) :467–480, 1977.

[25] Emiliano Lorini and François Schwarzentruber. A Modal Logic of Epistemic Games. *Games*, 1(4) :478–526, 2010.

[26] Fabio Massacci. Design and Results of the TABLEAUX-99 Non-classical (Modal) Systems Comparison. In *Proc. of Tableaux'99*, pages 14–18, 1999.

[27] Fabio Massacci and Francesco M. Donini. Design and Results of TANCS-2000 Non-classical (Modal) Systems Comparison. In *Proc. of Tableaux'00*, pages 52–56, 2000.

[28] Fabio Masssacci. Single step tableaux for modal logics : Methodology, computations, algorithms. *Journal of Autom.*, 24(3) :319–364, 2000.

[29] Alexandre Niveau and Bruno Zanuttini. Efficient Representations for the Modal Logic S5. In *Proc. of IJCAI'16*, pages 1223–1229, 2016.

[30] Peter F. Patel-Schneider and Roberto Sebastiani. A New General Method to Generate Random Modal Formulae for Testing Decision Procedures. *J. Artif. Intell. Res.*, 18 :351–389, 2003.

[31] Johan van Benthem Patrick Blackburn and Frank Wolter. *Handbook of Modal Logic*. Elsevier, 2007.

[32] John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[33] Roberto Sebastiani and Armando Tacchella. SAT Techniques for Modal and Description Logics. *Handbook of Satisfiability*, 185 :781–824, 2009.

[34] Roberto Sebastiani and Michele Vescovi. Automated Reasoning in Modal and Description Logics via SAT Encoding : the Case Study of K(m)/ALC-Satisfiability. *JAIR*, 35(1) :343, 2009.

[35] Roberto Sebastiani and Adolfo Villafiorita. SAT-Based Decision Procedures for Normal Modal Logics : A Theoretical Framework. In *Proc. of AIMSA'98*, pages 377–388, 1998.

[36] Dmitry Tishkovsky, Renate A. Schmidt, and Mohammad Khodadadi. The Tableau Prover Generator MetTeL2. In *Proc. of JELIA'12*, pages 492–495, 2012.

[37] G. S Tseytin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer, 1983.

[38] Tom Murphy VII, Karl Crary, and Robert Harper. Distributed Control Flow with Classical Modal Logic. In *Proc. of CSL'05*, pages 51–69, 2005.

[39] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS Version 3.5. In *Proc. of CADE'09*, pages 140–145, 2009.

Algorithme Efficace pour la Fouille de Séquences Fréquentes avec la Programmation par Contraintes

John O.R. Aoga[1,2] * **Tias Guns**[3] **Pierre Schaus**[1]

[1] ICTEAM, UCLouvain, Belgique; [2] ED-SDI, UAC, Bénin
[3] VUB et KULeuven, Belgique
{john.aoga,pierre.schaus}@uclouvain.be tias.guns@vub.ac.be

Résumé

La programmation par contraintes (CP) a séduit la communauté de fouille de données grâce à son approche hautement déclarative et sa flexibilité. Cependant, ces avantages n'offrent pas une garantie d'efficacité. En témoigne la littérature où les méthodes basées sur la CP n'ont toujours pas réussi à être aussi performante que les méthodes spécialisées. Dans ce papier publié au *ECML-PKDD'16* [1], nous montrons comment en combinant les techniques des deux mondes on peut arriver à une méthode très robuste et efficace en plus d'être modulaire et flexible pour résoudre les problèmes de fouille de séquences fréquentes. Notre approche, intitulée *PPIC*, est une contrainte globale, conçue sur les méthodes de projection de base de données. Cette contrainte utilise une structure de données *auto-backtraquante* (*trailing*) pour stocker et restaurer efficacement les bases de données projetées. Le calcul des supports et le filtrage reposent sur des améliorations algorithmiques utilisant des données pré-calculées. Des expériences détaillées montrent comment cette approche, pour la première fois, surpasse à la fois les méthodes basées sur la CP et celles spécialisées. Ainsi, le moindre qu'on puisse dire est que le tandem fouille de données et CP a encore de beaux jours devant lui.

Problème. Etant donnée $I = \{1, \ldots, N\}$ un ensemble de N symboles, une séquence est une liste ordonnée d'éléments de I. Une séquence $\alpha = \langle \alpha_1 \alpha_2 \ldots \alpha_m \rangle$ est une sous-sequence d'une autre séquence $s = \langle s_1 s_2 \ldots s_n \rangle$ (ou inversement s est une super-séquence de α), noté $\alpha \preceq s$, ssi *(i)* $m \leq n$ et *(ii)* il existe une liste d'entiers $(j_1, \ldots, j_m)$, telle que $1 \leq j_1 \ldots \leq j_m \leq n$ avec $s_{j_i} = \alpha_i$. Une base de données de séquences est un ensemble de séquences : $SDB = \{(sid_i, s) \forall i \in [1, |SDB|]\}$ où sid_i est l'identifiant de la i^{eme} séquence s. La Table 1a est un exemple de SDB. Le nombre de séquences de SDB qui sont super-séquences de la séquence α est appelé *support*. Le problème de la fouille de séquence fréquente (SPM) est la recherche de toutes les séquences α qui ont un support supérieur à un seuil θ donné.

Ce problème est très étudié et est à l'origine de beaucoup de méthodes spécialisées dont les plus efficaces sont *PrefixSpan* [5] et *cSPADE* [6]. La méthode *PrefixSpan* doit son efficacité aux bases de données projetées (BDP). En effet, une BDP, notée $SDB_{|\alpha}$, est une partition de la SDB originale par rapport à α. C'est l'ensemble des suffixes des séquences dans lesquelles on a pu trouver une première correspondance de α. Considérons par exemple $\alpha = \langle A \rangle$. Cette sous-séquence se retrouve dans les séquences 1,2 et 3 avec les suffixes respectifs tels que présentés dans la Table 1b-suffixes. Cette BDP peut être stockée efficacement en ne conservant qu'un pointeur sur les positions dans chaque séquence (Table 1b-pos). Remarquez que le support de α est égale à la taille de sa BDP. Quand l'on étend α avec le symbole B, la nouvelle BDP se définit incrémentalement : $SDB_{|\langle AB \rangle} = (SDB_{|\langle A \rangle})_{|\langle B \rangle}$. Ainsi, en faisant grandir progressivement une sous-séquence, en commençant par la séquence vide, on construit les BDP on en déduit les supports et on peut vérifier lesquelles sont fréquentes.

a) SDB		b) $SDB_{\|\langle A \rangle}$		c) $SDB_{\|\langle AB \rangle}$		d) $SDB_{\|\langle ABC \rangle}$	
sid	sequence	*pos*	suffixes	*pos*	suffixes	*pos*	suffixes
sid_1	$\langle ABCBC \rangle$	2	$\langle \_BCBC \rangle$	3	$\langle \_\_CBC \rangle$	1	$\langle \_\_\_BC \rangle$
sid_2	$\langle BABC \rangle$	3	$\langle \_\_BC \rangle$	4	$\langle \_\_\_C \rangle$	1	$\langle \_\_\_\_ \rangle$
sid_3	$\langle AB \rangle$	2	$\langle \_B \rangle$	3	$\langle \_\_ \rangle$		
sid_4	$\langle BCD \rangle$						

TABLE 1 – Exemples de SDB et de BDP. (« _ » pour un symbole supprimé ; *pos* : les positions stockées).

*Papier doctorant : John O.R. Aoga[1,2] est auteur principal.

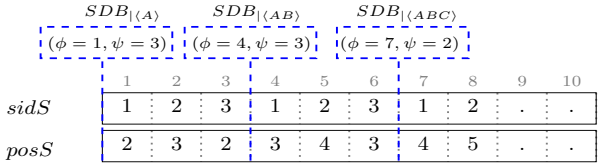


FIGURE 1 – Exemple de structure de données *auto-backtraquante* utilisant les BDP de la Table 1(-pos) .

En pratique, les solutions fréquentes générées sont souvent en grand nombre et ne sont pas forcément pertinentes. Afin d'affiner ou de superviser la fouille, d'autres contraintes sont souvent introduites. C'est là que les méthodes spécialisées (même les plus performantes) atteignent leur limite car manquant de flexibilité et de modularité. C'est ainsi, que les approches en CP ont été introduites [3, 2].

Objectif et contribution. Cependant, aucune de ces approches CP n'est aussi performante que les méthodes spécialisées. Ainsi, l'objectif de ce travail est de concevoir de nouvelles contraintes améliorant la littérature. La principale contribution de ce travail est la conception d'une *structure de données auto-backtraquante*.

Structure de données auto-backtraquante Le cœur de la méthode *PrefixSpan*, comme nous l'avons précédemment décrit, est la construction des BDP à chaque nœud de la recherche en profondeur. En effet, il faut remarquer que la BDP de α est utile pour construire l'extension de α à un symbole a donnée et en bactrackant la même BDP de α servira pour une nouvelle extension. Etant donnée qu'il est totalement inefficace de reconstruire les BDP à chaque *backtrack*, nous avons proposer de stocker et de restaurer les BDP en utilisant les techniques de *trailing* comme illustré à la Figure 1 (pour les valeurs voir la Table 1-pos) . On utilise deux vecteurs : $sidS$ et $posS$, pour maintenir respectivement les identifiants de séquences et la position dans chacune d'elles. Ces vecteurs sont des vecteurs « réversibles » : quand on étend une séquence, la BDP courante est lue. Une nouvelle est ensuite construite et stockée à sa suite en maintenant deux variables réversibles ϕ et ψ représentant la position courante dans les vecteurs et la taille de la BDP. Lors du *backtracking* les valeurs précédentes de ϕ et ψ sont restaurées et toutes celles après $\phi + \psi$ sont éventuellement écrasées. Cette structure permet non seulement d'optimiser l'utilisation de la mémoire mais rend plus rapide l'ensemble du processus de fouille. Plus encore, cette structure de données peut être utilisée dans n'importe quel problème utilisant la recherche en profondeur.

En utilisant cette structure de données et en la combinant avec des améliorations algorithmiques nous

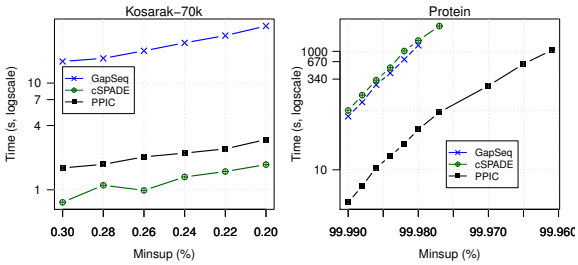


FIGURE 2 – Temps processeur pour `PPIC` avec *Gap-Seq* et *cSPADE* pour plusieurs θ (*minsup*).

avons proposés trois nouvelles *contraintes globales* dont la plus performante est *PPIC* (Prefix Projection Incremental Counting). De plus, ces contraintes peuvent être associées avec plusieurs contraintes additionnelles comme les contraintes sur la longueur des séquences, l'inclusion ou l'exclusion de certains symboles et les contraintes d'expression régulière.

Expériences et Résultats Comme illustré à la Figure 2, en utilisant deux bases de données (nombre de séquences/nombre de symboles) : l'une très éparse *Kosarak-70k* (69999/21144) et l'autre très dense *Protein* (103120/25), nous sommes clairement plus performant que *GapSeq* [2] sur tous les tableaux. Nous sommes aussi très compétitifs ou meilleurs que *cSPADE* [6]. C'est la toute première fois que les performances d'une approche CP dépassent à la fois celles des autres approches CP et spécialisées.

Nos approches sont implémentées en Scala dans le solveur OscaR [4]. Pour plus détail, le lecteur peut se référer au papier original [1] et/ou à notre site [1] où sont disponibles les données, le code et l'application.

Références

[1] J.O.R. Aoga, T. Guns, and P. Schaus. An efficient algorithm for mining frequent sequence with constraint programming. In *ECML PKDD 2016, Proceedings, Part II*. Springer International Publishing, 2016.

[2] A. Kemmar, S. Loudni, Y. Lebbah, P. Boizumault, and T. Charnois. A global constraint for mining sequential patterns with GAP constraint. In *CPAIOR 2016, Proceedings*, pages 198–215. Springer, 2016.

[3] B. Negrevergne and T. Guns. Constraint-based sequence mining using constraint programming. In *CPAIOR15, Proceedings*. Springer, 2015.

[4] OscaR Team. OscaR : Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`.

[5] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and MC. Hsu. Prefixspan : Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICCCN*, page 0215. IEEE, 2001.

[6] M.J. Zaki. Sequence mining in categorical domains : incorporating constraints. In *ICIKM*, pages 422–429. ACM, 2000.

1. `http://sites.uclouvain.be/cp4dm/spm/`

Comparaison de différents modèles de programmation par contraintes pour le clustering conceptuel

Maxime Chabert[1,3]*Pierre-Antoine Champin[2] Amélie Cordier[2] Christine Solnon[1]

[1] LIRIS, INSA Lyon, Lyon, France
[2] LIRIS, Université Lyon 1, Lyon, France
[3] Infologic, Bourg-lès-Valence, France
{prénom.nom}@liris.cnrs.fr

Résumé

Le clustering conceptuel permet de partitionner un ensemble d'objets en clusters d'objets similaires, correspondant à des concepts formels. Nous présentons une nouvelle approche basée sur la programmation par contraintes, où l'ensemble des concepts formels est extrait dans une étape de pré-traitement en utilisant des techniques spécialisées de fouille de données. Nous comparons l'efficacité de notre approche avec celle de plusieurs approches récentes utilisant la programmation par contraintes ou la programmation linéaire en nombres entiers sur des instances classiques d'apprentissage automatique. Nous introduisons également un nouvel ensemble d'instances provenant d'une application réelle, visant à extraire des concepts de paramétrage à partir de séquences de paramétrage d'un progiciel de gestion, et nous évaluons la pertinence des concepts extraits en fonction des critères utilisés dans la définition de la fonction objectif.

Abstract

Conceptual clustering allows to partition a set of objects into clusters of similar objects, corresponding to formal concepts. We present a new approach based on constraint programming where formal concepts are extracted in a pre-processing step by using a dedicated data-mining approach. We compare the efficiency of our approach with several recent approaches using constraint programming or integer linear programming on classical machine learning instances. We also introduce a new set of instances coming from a real application case, which aims at extracting setting concepts from an Enterprise Resource Planning (ERP) software. We assess the relevance of extracted concepts depending on criteria used in the objective function.

*Papier doctorant : Maxime Chabert[1,3] est auteur principal.

1 Introduction

Nous proposons dans cet article de nouveaux modèles à base de contraintes pour résoudre un problème de clustering conceptuel, et nous évaluons ces modèles sur des instances académiques, ainsi que sur un nouvel ensemble d'instances provenant d'une application réelle visant à automatiser la phase de paramétrage d'un progiciel de gestion (*Enterprise Ressource Planning*, ERP).

Présentation du contexte applicatif du travail. Les ERP sont des logiciels avec un vaste périmètre fonctionnel allant de la gestion commerciale jusqu'à la gestion des ateliers de production et de stockage [9]. Cette amplitude de fonctionnalités rend le processus d'installation complexe, et une étude récente souligne que 57% des installations d'ERP dépassent le budget et le temps prévus [1]. Nous avons étudié le processus d'installation du progiciel de gestion Copilote de la société Infologic, spécialisée en agro-alimentaire. Il apparaît que 65% du temps est dédié à la phase de paramétrage : cette phase consiste à affecter des valeurs à des paramètres afin de répondre aux besoins du client et à ses spécificités structurelles et organisationnelles [22]. Cette complexité est due au grand nombre de paramètres pouvant interagir entre eux. De plus, plusieurs études [17, 2] portant sur le processus d'installation des ERP montrent que le paramétrage n'est pas considéré comme un facteur critique de succès contrairement à l'accompagnement au changement ou à la formation des utilisateurs. C'est pourquoi, réduire le temps de paramétrage devient un réel enjeu pour les

intégrateurs d'ERP afin d'allouer plus de temps aux phases critiques du processus d'installation.

Pour répondre à cette problématique, nous proposons d'analyser une base de paramétrages existants, correspondant à des installations de l'ERP chez différents clients. Notre objectif est d'identifier des séquences pertinentes de paramétrage afin de les associer à des besoins fonctionnels. Étant donné que beaucoup de besoins se retrouvent chez plusieurs clients, ces séquences de paramétrage pourront être réutilisées durant l'installation de l'ERP chez un nouveau client ayant des besoins similaires.

Pour identifier des séquences pertinentes de paramétrage, nous proposons de partitionner la base de paramétrages de façon à regrouper les paramétrages similaires. Nous proposons pour cela d'utiliser le clustering conceptuel [15] car cette approche ne pré-suppose pas qu'il existe une fonction de similarité permettant d'évaluer la similarité de deux objets : chaque cluster correspond à un concept formel et est décrit par l'ensemble des paramètres communs à tous les paramétrages du cluster.

Présentation des contributions de l'article. Plusieurs travaux récents proposent de résoudre des problèmes de clustering conceptuel en utilisant des approches déclaratives telles que la programmation par contraintes (PPC) [4] ou la programmation linéaire en nombres entiers (PLNE) [18]. Ces approches sont particulièrement pertinentes dans notre contexte applicatif du fait de leur souplesse pour ajouter des contraintes ou modifier la fonction objectif : une de nos problématiques majeures est de trouver une fonction objectif et des contraintes permettant d'extraire des concepts de paramétrage pertinents pour les experts métiers.

Nous présentons une nouvelle approche basée sur la PPC. Comme proposé dans [18], nous introduisons une étape de pré-traitement pour extraire l'ensemble des concepts formels candidats à l'aide d'un outil dédié au problème de l'extraction de motifs fréquents. Nous proposons d'utiliser la programmation par contraintes pour sélectionner un sous-ensemble de cet ensemble formant une partition optimale, et nous introduisons deux nouveaux modèles ensemblistes pour cela. Nous comparons ces modèles avec ceux de [18] et [4]. Cette comparaison est réalisée sur un ensemble d'instances classiques dans le domaine de l'apprentissage automatique. Nous introduisons également un nouveau benchmark composé d'instances construites à partir de notre base de paramétrages. Enfin, nous comparons d'un point de vue qualitatif la qualité des clusterings obtenus selon différentes fonctions objectifs.

Organisation de l'article. La section 2 définit formellement le problème de clustering conceptuel, et décrit les approches permettant de résoudre ce problème, et plus particulièrement les approches récentes de [18] et [4]. La section 3 introduit deux nouveaux modèles PPC pour résoudre ce problème, l'un basé sur les clusters, l'autre sur les transactions. Enfin, la section 4 compare les différentes approches en termes de passage à l'échelle, et la section 5 compare la qualité des solutions calculées en fonction des critères considérés dans la fonction objectif.

2 Contexte

2.1 Clustering conceptuel

Le clustering conceptuel est une approche de classification non-supervisée qui vise à partitionner un ensemble d'objets en clusters homogènes. La particularité de cette méthode est qu'elle donne, en plus des clusters, une description de chaque cluster sous la forme d'un concept formel.

Soit $\mathcal{T}$ un ensemble de m transactions (ou objets), $\mathcal{I}$ un ensemble de n items (ou attributs), et $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{I}$ une relation binaire qui lie les transactions aux items : $(t, i) \in \mathcal{R}$ (noté également $t\mathcal{R}i$) traduit le fait qu'une transaction t contient l'item i. Nous supposons que toutes les transactions ont des ensembles d'items différents, *i.e.*,

$$\forall t, t' \in \mathcal{T}, t \neq t' \Rightarrow \{i \in \mathcal{I} : t\mathcal{R}i\} \neq \{i \in \mathcal{I} : t'\mathcal{R}i\}.$$

Étant donné un ensemble E, nous notons $\mathcal{P}(E)$ l'ensemble de ses sous-ensembles, et $\#E$ sa cardinalité. Enfin, sans perte de généralité, nous supposons que les transactions sont numérotées de 1 à m et les items de 1 à n.

L'*intention* d'un sous-ensemble $T \subseteq \mathcal{T}$ de transactions est l'ensemble des items contenus dans toutes les transactions de T, *i.e.*,

$$intent(T) = \{i \in \mathcal{I} : \forall t \in T, t\mathcal{R}i\}.$$

L'*extension* d'un sous-ensemble $I \subseteq \mathcal{I}$ d'items est l'ensemble des transactions qui contiennent tous les items de I, *i.e.*,

$$extent(I) = \{t \in \mathcal{T} : \forall i \in I, t\mathcal{R}i\}.$$

Ces deux opérateurs induisent une connexion de Galois entre $\mathcal{P}(\mathcal{T})$ et $\mathcal{P}(\mathcal{I})$, *i.e.*,

$$T \subseteq extent(I) \Leftrightarrow I \subseteq intent(T).$$

Un *concept formel* est un couple (T, I) avec $T \subseteq \mathcal{T}$ et $I \subseteq \mathcal{I}$ tels que $T = extent(I)$ et $I = intent(T)$. On

Table 1 – Jeu de données transactionnel $\mathcal{T}$

	i_1	i_2	i_3	i_4
t_1	1	0	0	1
t_2	1	0	1	1
t_3	0	1	0	1
t_4	0	1	1	0
t_5	1	0	1	0

note $\mathcal{F}$ l'ensemble de tous les concepts formels. Notons qu'un concept formel correspond à un ensemble clos d'items (*closed itemset*) tel que défini en fouille de données. Par conséquent, l'ensemble $\mathcal{F}$ des concepts formels peut être calculé en utilisant un algorithme de recherche de motifs clos fréquents (tel que LCM [23], par exemple), en fixant le seuil de fréquence à 1.

En clustering conceptuel, chaque cluster correspond à un concept formel, et un clustering est un ensemble de k concepts formels $\mathcal{C} = \{(T_1, I_1), \ldots, (T_k, I_k)\}$ tel que $\{T_1, \ldots, T_k\}$ forme une partition de l'ensemble de transactions $\mathcal{T}$.

La fréquence d'un cluster (T_j, I_j) est son nombre de transactions, *i.e.*, $freq(T_j, I_j) = \#T_j$, et sa taille est son nombre d'items, *i.e.*, $taille(T_j, I_j) = \#I_j$.

Différents critères peuvent être considérés pour définir la qualité d'un clustering conceptuel. Dans cet article, nous en considérons trois : (1) maximiser la taille minimale d'un cluster, de façon à éviter d'avoir des concepts comportant peu d'items ; (2) maximiser la fréquence minimale d'un cluster, de façon à éviter d'avoir des concepts comportant peu de transactions ; (3) maximiser la somme des tailles des clusters, de façon à favoriser les concepts comportant un grand nombre d'items.

Exemple. La table 1 présente un jeu de données transactionnelles $\mathcal{T}$ composé de cinq transactions définies sur quatre items. La table 2 donne l'ensemble $\mathcal{F}$ des concepts formels de $\mathcal{T}$. Par exemple, le concept c_1 est défini par le couple $(\{i_1\}, \{t_1, t_2, t_5\})$. Sa fréquence et sa taille sont : $freq(c_1) = 3$ et $taille(c_1) = 1$. $\mathcal{C}_1 = \{(\{i_1\}, \{t_1, t_2, t_5\}), (\{i_2\}, \{t_3, t_4\})\}$ et $\mathcal{C}_2 = \{(\{i_1\}, \{t_1, t_2, t_5\}), (\{i_2, i_4\}, \{t_3\}), (\{i_2, i_3\}, \{t_4\})\}$ sont deux exemples de clusterings de $\mathcal{T}$. Selon le critère considéré, la qualité de $\mathcal{C}_1$ (resp. $\mathcal{C}_2$) est évaluée à 1 (resp. 1), pour le critère de taille minimale d'un cluster, 2 (resp. 1), pour le critère de fréquence minimale d'un cluster, et 2 (resp. 5), pour le critère de somme des tailles des clusters.

2.2 Approches dédiées au clustering conceptuel

Depuis l'introduction du clustering conceptuel par [15], de multiples approches dédiées à ce problème ont été proposées.

Table 2 – Ensemble $\mathcal{F}$ des concepts formels de $\mathcal{T}$

C	*intent*	*extent*	*fréq.*	*taille*
c_1	$\{i_1\}$	$\{t_1, t_2, t_5\}$	3	1
c_2	$\{i_3\}$	$\{t_2, t_4, t_5\}$	3	1
c_3	$\{i_1, i_3\}$	$\{t_2, t_5\}$	2	2
c_4	$\{i_4\}$	$\{t_1, t_2, t_3\}$	3	1
c_5	$\{i_1, i_4\}$	$\{t_1, t_2\}$	2	2
c_6	$\{i_1, i_3, i_4\}$	$\{t_2\}$	1	3
c_7	$\{i_2\}$	$\{t_3, t_4\}$	2	1
c_8	$\{i_2, i_3\}$	$\{t_4\}$	1	2
c_9	$\{i_2, i_4\}$	$\{t_3\}$	1	2

Plusieurs de ces approches utilisent des heuristiques basées sur des mesures statistiques pour construire des clusters [8] pouvant être organisés en hiérarchie [5, 11]. Le système COBWEB [5] s'appuie sur la similarité intra-cluster et la dissimilarité inter-cluster pour construire incrémentalement une hiérarchie de clusters, l'interprétation conceptuelle des clusters étant alors une étape indépendante de la construction des clusters. Ces deux tâches sont souvent découplées dans des approches plus récentes soit en utilisant des techniques de clustering après avoir extrait un ensemble de concepts [19], soit en clusterisant des objets puis en extrayant une description associée à chaque cluster [20]. D'autres approches ont introduit l'utilisation de connaissances lors de la construction des clusters pour améliorer la pertinence des concepts extraits [16, 10, 24].

La qualité des résultats obtenus par ces approches reste variable tout comme le passage à l'échelle sur de grands volumes de données.

Ces approches dédiées ne permettent pas facilement d'ajouter de nouvelles contraintes, ou de modifier la fonction objectif. Ce point étant particulièrement important dans notre contexte applicatif où nous souhaitons évaluer la qualité de différents clusterings obtenus en considérant différents critères, nous nous sommes intéressés à des approches déclaratives telles que la PPC et la PLNE.

2.3 PPC pour la recherche d'ensembles d'items clos

L'utilisation de la PPC pour modéliser et résoudre des problèmes de recherche d'ensembles d'items clos est un sujet communément explorée durant les dix dernières années [21, 12, 7]. Plus récemment, Lazaar et al. [14] ont introduit une contrainte globale pour extraire des ensemble d'items clos fréquents. Cette contrainte globale assurant la consistance du domaine en un temps polynomial est un ordre de grandeur plus

lent que LCM [23] mais peut devenir plus performant pour des requêtes plus complexes où des contraintes supplémentaires sont ajoutées.

2.4 PPC pour le clustering conceptuel

Guns a montré dans sa thèse [6] que la PPC fournit un cadre déclaratif permettant de facilement modéliser différents problèmes de recherche de motifs fréquents, et que les solveurs génériques de PPC peuvent être compétitifs avec des algorithmes dédiés. Guns a notamment proposé une modélisation utilisant des variables binaires pour exprimer le fait qu'un item appartient à l'intention d'un concept associé à un cluster. Dao et al [4] ont proposé un nouveau modèle utilisant des variables ensemblistes, et ont montré que ce modèle ensembliste a de bien meilleures performances en pratique que le modèle binaire. Nous décrivons ici ce modèle ensembliste.

Variables. Pour chaque transaction $t \in \mathcal{T}$, la variable entière G_t représente le cluster de t. Le nombre de clusters est défini par une constante k donnée en entrée, et les clusters sont numérotés de 1 à k. Ainsi, chaque variable G_t a pour domaine $D(G_t) = [1, k]$.

Pour chaque cluster $c \in [1, k]$, la variable ensembliste E_c représente l'ensemble des items de l'intention du concept formel associé au cluster c. Le domaine de E_c est l'ensemble des sous-ensembles de $\mathcal{I}$, *i.e.*, $D(E_c) = \mathcal{P}(\mathcal{I})$.

Contraintes. Les symétries (dues au fait que les clusters sont interchangeables) sont éliminées en posant une contrainte de précédence [13] :

$$precede(G, [1, k]).$$

Cette contrainte assure que la première transaction appartient au premier cluster (*i.e.*, $G_1 = 1$), et que $\forall j \in [2, n], \exists l < j, G_l = G_j - 1$.

Chaque cluster est contraint à posséder au moins une transaction à l'aide de la contrainte :

$$atLeast(1, G, k).$$

La contrainte d'extension est exprimée par :

$$\forall c \in [1, k], \forall t \in \mathcal{T}, G_t = c \Leftrightarrow E_c \subseteq \{i \in \mathcal{I} | t \mathcal{R} i\}.$$

La contrainte d'intention est exprimée par :

$$\forall c \in [1, k], E_c = \cap_{t \in \mathcal{T}, G_t = c} \{i \in \mathcal{I} | t \mathcal{R} i\}.$$

Chaque contrainte d'intention nécessite n contraintes de domaine réifiées pour construire l'ensemble $I_c = \{t \in \mathcal{T} | G_t = c\}$, et une contrainte *element* ensembliste.

Fonction objectif. Pour maximiser la fréquence minimale des clusters, on introduit une variable entière F devant être maximisée. Son domaine est $D(F) = [1, m]$, et elle est contrainte à être inférieure ou égale à la fréquence de chaque cluster $c \in [1, k]$ en posant une contrainte $atLeast(F, G, c)$.

Pour maximiser la taille minimale des clusters, on introduit une variable entière T devant être maximisée. Son domaine est $D(T) = [1, n]$, et elle est contrainte à être inférieure ou égale à la taille de chaque cluster $c \in [1, k]$ en posant la contrainte $T \leq \#E_c$.

Si le cas n'a pas été explicitement étudié dans [4], on peut facilement étendre ce modèle pour maximiser la somme des tailles en ajoutant une variable S devant être maximisée. Son domaine est $D(S) = nk$, et elle est contrainte à être égale à la somme des variables T.

Extension du modèle à un nombre variable de clusters. Le modèle introduit dans [4] suppose que le nombre de clusters est fixé par une constante k. L'extension au cas où le nombre de clusters n'est pas connu *a priori* est relativement triviale : il suffit d'introduire une constante $kMax$, fixant le nombre maximal de clusters (si le nombre de clusters n'est pas borné, alors $kMax = m$), et de définir k comme une variable entière de domaine $D(k) = [2, kMax]$. Dans ce cas, la contrainte $atLeast(1, G, k)$ n'a plus de raison d'être.

2.5 PLNE pour le clustering conceptuel

Ouali et al [18] ont proposé de combiner un outil dédié à l'extraction de motifs fréquents avec la PLNE pour faire du clustering conceptuel : dans une étape de pré-traitement, l'ensemble de tous les concepts formels est calculé en utilisant un outil dédié à ce problème tel que LCM [23] ; la PLNE est utilisée ensuite pour sélectionner un sous-ensemble de ces concepts qui forme une partition de $\mathcal{T}$ et qui optimise la fonction objectif.

Plus précisément, soit $\mathcal{F}$ l'ensemble de tous les concepts formels calculés en pré-traitement. L'objectif est de sélectionner un sous-ensemble de $\mathcal{F}$ tel que chaque transaction de $\mathcal{T}$ appartienne à exactement un concept formel du sous-ensemble, et optimise un critère donné. Cela est modélisé en PLNE dans [18] de la façon suivante.

Variables. Pour chaque concept formel $f \in \mathcal{F}$, on introduit une variable binaire x_f telle que $x_f = 1$ ssi le concept formel f est sélectionné.

La variable entière k correspond au nombre de concepts sélectionnés.

Contraintes. Pour garantir que l'ensemble des concepts sélectionnés forme une partition de $\mathcal{T}$ on

pose, pour chaque transaction $t \in \mathcal{T}$, la contrainte :

$$\sum_{f \in \mathcal{F}} a_{tf} x_f = 1.$$

où $a_{tf} = 1$ si la transaction t appartient à l'extension du concept f.

Pour contraindre k à être égal au nombre de concepts sélectionnés, on pose la contrainte :

$$k = \sum_{f \in \mathcal{F}} x_f.$$

Enfin, on peut borner le nombre de concepts sélectionnés k en posant la contrainte :

$$kMin \leq k \leq kMax.$$

Fonction objectif. Un gain v_f est associé à chaque concept formel $f \in \mathcal{F}$: v_f est égal à la taille de f. La fonction objectif à maximiser est la somme des gains :

$$\sum_{f \in \mathcal{F}} v_f x_f.$$

Si le cas n'a pas été explicitement étudié dans [18], on peut facilement étendre le modèle pour maximiser le gain minimal d'un concept. Le gain v_f est égal soit à la taille de f, soit à sa fréquence, selon le critère choisi. Une variable v_{min} est introduite et est contrainte à être inférieure ou égale au gain des concepts sélectionnés en posant, pour chaque concept formel $f \in \mathcal{F}$, la contrainte :

$$v_{min} \leq v_f x_f + M(1 - x_f)$$

où M est une constante positive supérieure au plus grand gain possible. La fonction objectif à maximiser est v_{min}.

3 Nouveaux modèles PPC

Nous proposons de nous inspirer de l'approche de [18], consistant à extraire dans une phase de prétraitement l'ensemble $\mathcal{F}$ de tous les concepts formels avec un outil dédié à ce problème, et nous proposons d'évaluer les capacités de la programmation par contraintes pour sélectionner le sous-ensemble de concepts formels formant un clustering optimal.

Nous proposons deux modèles utilisant des contraintes ensemblistes : le premier modèle associe une variable entière à chaque cluster (déterminant le concept formel associé au cluster) et pose une contrainte ensembliste de partition sur l'ensemble des extensions des concepts formels associés aux clusters ; le second modèle utilise une variable ensembliste pour représenter le sous-ensemble de clusters sélectionnés et pose des contraintes *member* et *card* pour assurer que ce sous-ensemble définit bien une partition.

3.1 Modèle ensembliste basé sur les clusters

Variables. Pour chaque cluster $c \in [1, kMax]$, on définit une variable entière G_c déterminant le concept formel associé au cluster c. Comme la solution optimale peut avoir moins de $kMax$ clusters, on introduit un concept formel vide : ce concept a le numéro 0, et son extension est l'ensemble vide, *i.e.*, $extent(0) = \emptyset$. Nous supposons que les concepts formels de $\mathcal{F}$ sont numérotés de 1 à p. Par conséquent, le domaine de chaque variable G_c est $D(G_c) = [0, p]$: si $G_c \in [1, p]$, alors le cluster c correspond au concept formel G_c ; si $G_c = 0$ alors le cluster c est vide.

On introduit une variable entière k qui représente le nombre de clusters non vides de la solution, et une autre variable entière k_{empty} qui représente le nombre de clusters vides de la solution. Leurs domaines sont, respectivement, $D(k) = [2, kMax]$ et $D(k_{empty}) = [0, kMax - 2]$.

Contraintes. Pour éliminer les symétries, dues au fait que les valeurs affectées à deux variables G_i et G_j peuvent être interchangées, nous contraignons G à prendre des valeurs croissantes par rapport à un ordre défini sur $\mathcal{F}$. Pour assurer que les extensions des clusters forment une partition de l'ensemble des transactions, nous posons une contrainte de partition [3] :

$$partition(\{extent(G_c) | c \in [1, kMax]\}, [1, m])$$

Pour assurer que le nombre de clusters vides est égal à k_{empty}, nous posons la contrainte :

$$count(G, 0, k_{empty})$$

Enfin, nous assurons que k est égal au nombre de clusters non vides à l'aide de la contrainte :

$$k + k_{empty} = kMax.$$

Fonction objectif. Un gain v_f est associé à chaque concept formel $f \in \mathcal{F}$. Selon les cas, ce gain peut être la fréquence ou la taille de f. La fonction objectif à maximiser peut être soit la somme des gains, *i.e.*,

$$\sum_{c=1}^{kMax} v_{G_c}$$

(et dans ce cas on définit le gain du cluster vide par $v_0 = 0$), soit le gain minimal, *i.e.*,

$$\min_{c \in [1, kMax]} v_{G_c}$$

(et dans ce cas on définit le gain du cluster vide par $v_0 = \infty$).

Stratégie de recherche. Deux stratégies de recherches différentes sont utilisées selon si la taille ou la fréquence est utilisée comme critère de la fonction objectif. Dans les deux cas, les concepts formels sont classés par ordre décroissant selon leur gain v_f. Ainsi, $\forall f, f' \in F, f > f' \Leftrightarrow v_f < v_{f'}$.

Si le critère considéré est la taille, les solutions tendent à avoir un nombre de clusters proche de $kMax$. Nous utilisons un sélecteur qui choisit la borne minimale du domaine comme prochaine valeur pour la variable k_{empty}. De la même manière, pour chaque variable G_c, les concepts formels de plus grande taille sont d'abord choisis. Ainsi, les solutions avec un maximum de clusters non vides sont recherchées en premier.

Si le critère considéré est la fréquence, les solutions tendent à avoir peu de clusters. Les variables de décision sont uniquement les variables G_c. Nous utilisons la stratégie *first fail* qui consiste à sélectionner la variable avec le plus petit domaine comme prochaine variable à instancier. De plus, le sélecteur des variables G_c choisit la borne minimale du domaine comme prochaine valeur. Ainsi, les solutions avec le moins de clusters possibles sont explorées en premier.

3.2 Modèle ensembliste basé sur les transactions

Variables. Pour chaque transaction $t \in \mathcal{T}$, nous définissons une variable entière C_t déterminant le concept sélectionné dont l'extension contient t : chaque transaction t doit appartenir à l'extension d'exactement un concept sélectionné, et l'ensemble des concepts candidats est l'ensemble des concepts dont l'extension contient t. Ainsi, pour chaque transaction $t \in \mathcal{T}$, le domaine de C_t est $D(C_t) = \{f \in F \mid t \in extent(f)\}$.

Nous définissons une variable ensembliste P déterminant l'ensemble des concepts sélectionnés : chaque concept appartenant à P correspond à un cluster. Le domaine de P est : $D(P) = \mathcal{P}(\mathcal{F})$.

La variable entière k définit le nombre de clusters de la solution (et donc la cardinalité de P). Son domaine est : $D(k) = [2, kMax]$.

Contraintes. On assure que, pour chaque transaction $t \in \mathcal{T}$, C_t est un élément de l'ensemble P en posant la contrainte :

$$member(C_t, P)$$

On assure que, pour chaque transaction $t \in \mathcal{T}$, il y a exactement un concept formel de P dont l'extension contient t (autrement dit, les extensions des concepts de P forment une partition de l'ensemble des transactions) en vérifiant que l'intersection entre P et l'ensemble des concepts dont t appartient à l'extension contient un seul élément, *i.e.*,

$$\forall t \in \mathcal{T}, card(\{f \in F \mid t \in extent(f)\} \cap P) = 1$$

Enfin, le nombre de clusters de la solution est contraint avec la contrainte $card(P) = k$, assurant que le nombre d'éléments de P est égal à k, *i.e.*, le nombre de clusters est égal à k.

Fonction objectif. Comme pour le modèle précédent, la fonction objectif est définie en associant un gain v_f à chaque concept formel f.

Stratégie de recherche. Les concepts formels sont classés par ordre décroissant selon leur gain v_f. Les variables de décision sont uniquement les variables C_t avec une stratégie *first fail*. De plus, le sélecteur des variables C_t choisit la borne minimale du domaine comme prochaine valeur.

4 Comparaison expérimentale des différents modèles

Dans cette section, nous comparons l'efficacité de nos modèles par rapport aux approches de Ouali et al. [18] et de Dao et. al [4].

Protocole expérimental. Toutes les expérimentations ont été menées sur un Intel(R) Core(TM) i7-6700 avec 3.40GHz de CPU et 65GB de RAM. Nous avons utilisé LCM [23] pour extraire les concepts formels, Gecode v4.3 pour les modèles PPC et Cplex v12.7 pour le modèle PLNE. Dans chaque modèle, nous avons fixé $kMax$, le nombre de cluster maximal, à $m-1$, m étant le nombre de transactions de l'instance. Chaque résolution a été limitée à deux heures de temps CPU.

Description des instances. Nous avons considéré quatre instances classiques en apprentissage automatique et utilisées dans [18] : zoo, vote, tic-tac-toe et

Table 3 – Description des jeux de données : chaque ligne donne successivement le nom du jeu de données, le nombre de transactions, le nombre d'items, la densité, le nombre de concepts, et le temps (en secondes) mis par LCM pour extraire les concepts.

Nom	$\#\ \mathcal{T}$	$\#\ \mathcal{I}$	$\rho(\%)$	$\#\ \mathcal{F}$	Temps
ERP 1	50	27	48	1 580	0,01
ERP 2	84	42	45	14 305	0,05
ERP 3	95	61	48	71 918	0,45
ERP 4	160	66	45	728 537	5,31
zoo	59	36	44	4 567	0,01
vote	341	48	34	227 031	0,54
tic-tac-toe	958	27	33	42 711	0,05
mushroom	8 124	119	19	221 524	3,85

TABLE 4 – Comparaison des temps de résolution : chaque ligne donne successivement le nom de l'instance, et les résultats pour les trois critères à maximiser (taille minimale, fréquence minimale, et somme des tailles). Pour chaque critère, nous donnons les temps CPU (en secondes) des trois modèles PPC (FullCP, CB et TB) et du modèle PLNE (LP). Pour les modèles CB, TB et LP, le temps CPU comprend le temps mis par LCM pour extraire les concepts formels. Le symbole "-" indique que le temps dépasse la limite de 2h.

Instance	(1) - Max. taille minimale				(2) - Max. fréquence minimale				(3) - Max. somme des tailles			
	FullCP	CB	TB	LP	FullCP	CB	TB	LP	FullCP	CB	TB	LP
ERP 1	0,6	0,3	**0,0**	0,4	**0,1**	0,2	**0,1**	0,8	-	-	-	**0,1**
ERP 2	6,3	4,8	**0,4**	18,3	3,5	6,1	**0,6**	13,6	-	-	-	**1,7**
ERP 3	5,9	24,0	**2,6**	1 722,7	12,0	282,6	**6,7**	143,3	-	-	-	**14,0**
ERP 4	74,6	457,9	**35,6**	-	1 613,2	-	**204,6**	6833,7	-	-	-	**183,2**
zoo	1,7	0,8	**0,1**	2,0	0,4	0,4	**0,1**	1,5	-	-	-	**0,3**
vote	2885,6	1478,0	**46,6**	4312,8	-	-	**10,2**	55,2	-	-	-	**51,9**
tic-tac-toe	-	-	**240,4**	254,5	1 115,1	5 982,0	**10,1**	718,6	-	-	-	**32,0**
mushroom	-	-	-	-	-	-	**1 577,5**	-	-	-	-	**1635,1**

mushroom. Nous avons également considéré quatre instances construites à partir de notre base de paramétrages. Cette base comporte 400 paramétrages, chaque paramétrage correspondant à une installation de l'ERP Copilote chez un client différent. Chacun de ces paramétrages spécifie les valeurs de près de 450 paramètres (chaque paramètre pouvant prendre un nombre fini de valeurs différentes). Nous avons transformé chaque couple paramètre/valeur en un item booléen et extrait quatre instances de tailles différentes pour pouvoir évaluer plus finement le passage à l'échelle des différentes approches considérées. La table 3 présente les caractéristiques de chaque instance ainsi que le temps d'extraction des concepts formels avec LCM. Nous pouvons remarquer que ce temps dépend du nombre de concepts formels (la complexité de LCM est linéaire par rapport à $\#\mathcal{F}$), et est relativement court. Par exemple, les 728 537 concepts formels de l'instance ERP 4 sont extraits en moins de six secondes.

Modèles comparés. Nous avons comparé les deux modèles décrits dans la partie 2.2, à savoir l'approche de Ouali et al. [18] (appelée LP) et celle de Dao et al. [4] (appelée FullCP), avec nos deux modèles introduits dans la partie 3 : le modèle basé sur les clusters (appelé CB) et celui basé sur les transactions (appelé TB).

La table 4 compare les temps de résolution des différents modèles pour les trois critères d'optimisation considérés. Pour les modèles CB, TB et LP, le temps CPU comprend le temps de pré-traitement (*i.e.*, le temps mis par LCM pour extraire les concepts formels).

Temps de résolution pour les critères (1) et (2). Quand le critère considéré est la maximisation du minimum de la taille (1) ou de la fréquence (2), le modèle TB domine tous les autres modèles sur l'ensemble des instances. Il est souvent un ordre de grandeur plus rapide que les autres modèles. Cependant, il ne trouve pas la solution optimale dans le temps imparti pour l'instance mushroom pour le minimum de la taille (1). Cela s'explique probablement par le nombre de transactions plus important (10^3) comparé aux autres instances (de 10^1 à 10^2).

La différence de performance du modèle TB avec le modèle CB est probablement due à l'efficacité de la stratégie *first fail* qui permet de choisir la variable C_t ayant le moins de concepts formels dans son domaine. Dans les deux modèles, l'espace des concepts candidats (*i.e.*, les concepts dont l'extension n'a aucune transaction appartenant aux extensions des concepts déjà sélectionnés) est réduit de la même manière par la contrainte de partition. En revanche, le modèle TB choisit en priorité les concepts contenus dans la transaction ayant le moins de candidats (contrairement au modèle CB), et réduit ainsi efficacement l'espace de recherche pour converger plus rapidement vers la solution optimale. De plus, l'heuristique de CB sur le nombre de clusters vides n'est pas toujours efficace, comme par exemple pour tic-tac-toe avec le critère (2).

Nous pouvons noter les bonnes performances de FullCP sur les instances ERP : FullCP est capable de résoudre toutes ces instances (pour les critères (1) et (2)), et il est souvent plus rapide que CB et LP. En revanche, il n'est capable de résoudre que la moitié des quatre instances académiques.

Enfin, le modèle LP est toujours moins efficace que TB et ne parvient pas à résoudre deux instances pour les critères (1) (ERP 4 et mushroom) et (2) (vote et mushroom). Le plus grand nombre de concepts formels de ces instances (10^5 contre 10^4 et 10^3) explique probablement ces performances.

TABLE 5 – Temps CPU (en secondes) pour trouver la solution optimale avec le modèle TB quand le critère est la maximisation de la somme des tailles (3).

Instance	Temps
ERP 1	0,1
ERP 2	1,1
ERP 3	9,4
ERP 4	815,8
zoo	0,3
vote	831,2
tic-tac-toe	930,4

Temps de résolution pour le critère (3). Quand le critère considéré est la maximisation de la somme des tailles, la seule approche capable de résoudre des instances est LP. Les approches PPC ne résolvent aucune instance en moins de deux heures, alors que certaines instances (ERP 1 et zoo) sont résolues en moins d'une seconde par la PLNE.

Cependant, nous avons constaté que le modèle TB trouve très rapidement la solution optimale. De fait, pour les huit instances considérées, la première solution trouvée par TB est la solution optimale. Les temps mis pour trouver cette solution sont donnés dans la table 5. Ces temps sont inférieurs à 10 secondes pour ERP 1, ERP 2, ERP 3 et zoo, et ils sont inférieurs à 1000 secondes pour ERP 4, tic-tac-toe et vote. Ainsi, TB trouve relativement rapidement la solution optimale mais n'est pas capable de prouver l'optimalité dans la limite de deux heures. Cela provient probablement des heuristiques de choix considérées, qui permettent de guider la recherche vers les bonnes solutions mais ne sont pas efficaces pour prouver l'optimalité.

5 Comparaison de la qualité des solutions en fonction des critères

Dans cette section, nous comparons les clusterings calculés en fonction des critères considérés dans la fonction objectif.

Mesures de performance. Pour évaluer la qualité des clusterings obtenus, nous avons utilisé deux mesures classiques de performance, *i.e.*, la similarité intra-cluster (ICS) et la dissimilarité inter-cluster (ICD). La similarité entre deux transactions est définie par la fonction $s : \mathcal{T} \times \mathcal{T} \rightarrow [0, 1]$ telle que $s(t, t')$ correspond au ratio entre la taille de l'intersection des items de t et t' et la taille de leur union :

$$s(t, t') = \frac{\#\{i \in \mathcal{I} : t\mathcal{R}i \wedge t'\mathcal{R}i\}}{\#\{i \in \mathcal{I} : t\mathcal{R}i \vee t'\mathcal{R}i\}}$$

ICS est la similarité moyenne des paires de transactions appartenant à un même cluster :

$$ICS(C_1, \ldots, C_k) = \frac{1}{2} \sum_{i=1}^{k} \Big(\sum_{t,t' \in C_i} (s(t, t')) \Big)$$

Plus ICS est proche de 1, et plus les clusters sont homogènes (*i.e.*, deux transactions à l'intérieur d'un même cluster partagent une grande proportion d'items). Évidemment, cette mesure doit être mise en perspective avec le nombre de transactions dans chaque cluster : une partition où chaque cluster comporte une seule transaction a une valeur ICS égale à 1.

ICD est la dissimilarité moyenne des paires de transaction appartenant à des clusters différents :

$$ICD(C_1, \ldots, C_k) = \sum_{1 \leq i < j \leq k} \Big(\sum_{t \in T_{C_i}, t' \in T_{C_j}} (1 - s(t, t')) \Big)$$

Plus ICD est proche de 1, et plus les clusters sont bien séparés (*i.e.*, deux transactions appartenant à des clusters différents partagent peu d'items). Là encore, cette mesure doit être mise en perspective avec le nombre de clusters : une partition comportant un seul cluster a une valeur ICD égale à 1.

La table 6 donne, pour chaque instance, le nombre k de clusters ainsi que les valeurs ICS et ICD des clusterings optimaux selon chacun des trois critères considérés.

Évaluation pour les critères liés à la taille. Considérons tout d'abord les résultats obtenus lorsque le critère à optimiser est lié à la taille des clusters : (1) maximiser la taille minimale ou (3) maximiser la somme des tailles. Dans ce cas, la fusion de deux clusters ne peut que dégrader la qualité de la solution, et donc les solutions optimales ont tendance à avoir un grand nombre de clusters. En pratique, nous observons que pour toutes les instances considérées, le nombre k de clusters dans la solution optimale pour les critères (1) et (3) est égal à $kMax = m - 1$. Autrement dit, tous les clusters ont une seule transaction sauf un qui en a deux. Cela vient du fait que, pour toutes les instances, chaque transaction est un concept formel (autrement dit, pour chaque transaction t_j, il n'existe pas de transaction t_k telle que l'ensemble des items de t_j soit strictement inclus dans l'ensemble des items de t_k, et donc $extent(intent(\{t_j\})) = \{t_j\}$). La conséquence immédiate du fait que les solutions optimales pour les critères (1) et (3) ont $m - 1$ clusters est que ICS est très proche de 1. Nous observons que pour toutes les instances sauf deux (ERP 3 et zoo) les solutions calculées avec les critères (1) et (3) ont des ICS et ICD identiques. Pour ERP 3 et zoo, ICS et ICD sont très proches.

TABLE 6 – Comparaison de la qualité des clusters : chaque ligne donne successivement le nom de l'instance, et la valeur de k, ICS et ICD de la solution optimale pour chacun des trois critères considérés ((1) - maximiser la taille minimale, (2) - maximiser la fréquence minimale, et (3) - maximiser la somme des tailles).

Instance	(1) - taille min.			(2) - fréq. min.			(3) - somme tailles		
	k	ICS	ICD	k	ICS	ICD	k	ICS	ICD
ERP 1	49	**0,9971**	0,4956	2	0,5494	**0,5409**	49	**0,9971**	0,4956
ERP 2	83	**0,9988**	0,3769	2	0,6719	**0,4220**	83	**0,9988**	0,3769
ERP 3	94	0,9966	0,3429	2	0,7242	**0,3876**	94	**0,9993**	0,3311
ERP 4	159	**0,9996**	0,3424	2	0,7072	**0,3945**	159	**0,9996**	0,3424
zoo	58	**0.9980**	0.5724	2	0,4912	**0,5937**	58	0,9945	0,5709
tic tac toe	957	**0,9996**	0,7724	3	0,2919	**0,8042**	957	**0,9996**	0,7724
vote	340	**0,9997**	0,6720	3	0,4279	**0,7277**	340	**0,9997**	0,6720
mushroom	-	-	-	2	0,3793	0,7671	-	-	-

Évaluation pour le critère lié à la fréquence. Considérons maintenant les résultats obtenus lorsque le critère à optimiser est (2) maximiser la fréquence minimale. Dans ce cas, l'éclatement d'un cluster en deux clusters ne peut que dégrader la qualité de la solution, et donc les solutions optimales ont tendance à avoir un petit nombre de clusters. En pratique, toutes les solutions optimales ont 2 clusters, sauf tic-tac-toe et vote qui ont 3 clusters. Dans ce cas, ICS est nettement inférieure à 1 : en moyenne, les transactions d'un même cluster partagent entre 29% (pour tic-tac-toe) et 72% (pour ERP 3) d'items. En contrepartie, ICD est plus importante que pour les critères (1) et (3) qui privilégient la taille : pour le critère (2), la dissimilarité moyenne de transactions de deux clusters différents varie entre 39% (pour ERP 3 et ERP 4) et 80% (pour tic-tac-toe), tandis que pour les critères (1) et (3), elle varie entre 33% ou 34% (pour ERP 3 et ERP 4) et 77% (pour tic-tac-toe).

6 Conclusion

Nous avons proposé une nouvelle approche de clustering conceptuel basée sur la programmation par contraintes où l'ensemble des concepts formels est extrait dans une étape de pré-traitement. L'évaluation expérimentale montre que notre approche est plus efficace en comparaison avec des travaux récents pour des fonctions objectifs maximisant un minimum, obtenant des clusterings de qualité similaire à ceux obtenus en maximisant la somme des tailles des clusters.

Dans cette première série d'expérimentations, nous n'avons pas introduit de contraintes liées à notre application. En particulier, le nombre k de clusters n'est pas contraint et peut prendre n'importe quelle valeur comprise entre 2 et $m-1$. Les premiers résultats montrent que cela ne permet pas d'extraire des clusters à haute valeur ajoutée pour les experts : pour les critères (1) et (3) qui privilégient la taille des clusters, ils sont trop nombreux et spécialisés, tandis que pour le critère (2) qui privilégie la fréquence, ils sont trop peu nombreux et généraux.

Aussi prévoyons-nous dans la suite de nos travaux de rechercher des clusterings plus pertinents en explorant plusieurs pistes. Tout d'abord, nous proposons d'utiliser notre procédure pour extraire les clusterings non dominés selon les critères (1) et (2) pour avoir un panel plus diversifié de clusterings, notamment sur le nombre k de clusters. L'utilisation de nouvelles mesures comme la diversité qui est utilisée dans [18] ou l'écart entre la fréquence maximale et minimale sont à expérimenter également. Par ailleurs, nous proposons d'appliquer itérativement notre procédure de clustering afin de calculer une hiérarchie de clusterings en adoptant soit une démarche descendante (partitionner progressivement les clusters en partant d'un unique cluster comportant toutes les transactions) ou ascendante (fusionner progressivement les clusters en partant d'une partition comportant un cluster par transaction).

Références

[1] Solutions, p.c. : Panorama consulting solutions research report - 2016 erp report, 2016.

[2] M. Munir Ahmad and Ruben Pinedo Cuenca. Critical success factors for erp implementation in smes. *Robot. Comput.-Integr. Manuf.*, 29(3) :104–111, June 2013.

[3] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. *Disjoint, Partition and Intersection Constraints for Set and Multiset Variables*, pages 138–152. Springer Berlin Heidelberg, 2004.

[4] Thi-Bich-Hanh Dao, Willy Lesaint, and Christel Vrain. Clustering conceptuel et relationnel en programmation par contraintes. In *JFPC 2015*, Bordeaux, France, June 2015.

[5] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Mach. Learn.*, 2(2) :139–172, September 1987.

[6] Tias Guns. Declarative pattern mining using constraint programming. *Constraints*, 20(4) :492–493, 2015.

[7] Tias Guns, Siegfried Nijssen, and Luc De Raedt. Itemset mining : A constraint programming perspective. *Artif. Intell.*, 175(12-13) :1951–1983, 2011.

[8] Stephen José Hanson and Malcolm Bauer. Conceptual clustering, categorization, and polymorphy. *Machine Learning*, 3(4) :343–372, 1989.

[9] L. Hossain. *Enterprise Resource Planning : Global Opportunities and Challenges : Global Opportunities and Challenges*. IRM Press, 2001.

[10] A. Hotho and G. Stumme. Conceptual clustering of text clusters. In G. Kókai and J. Zeidler, editors, *Proc. Fachgruppentreffen Maschinelles Lernen (FGML 2002)*, pages 37–45, 2002.

[11] Istvan Jonyer, Lawrence B. Holder, and Diane J. Cook. Graph-based hierarchical conceptual clustering in structural databases. In Henry A. Kautz and Bruce W. Porter, editors, *AAAI/IAAI*, page 1078. AAAI Press / The MIT Press, 2000.

[12] Mehdi Khiari, Patrice Boizumault, and Bruno Crémilleux. Constraint programming for mining n-ary patterns. In *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, pages 552–567, 2010.

[13] Yat Chiu Law and Jimmy H. M. Lee. *Global Constraints for Integer and Set Value Precedence*, pages 362–376. Springer Berlin Heidelberg, 2004.

[14] Nadjib Lazaar, Yahia Lebbah, Samir Loudni, Mehdi Maamar, Valentin Lemière, Christian Bessiere, and Patrice Boizumault. A global constraint for closed frequent pattern mining. In *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, pages 333–349, 2016.

[15] R.S. Michalski. *Knowledge Acquisition Through Conceptual Clustering : A Theoretical Framework and an Algorithm for Partitioning Data Into Conjunctive Concepts*. Report (University of Illinois at Urbana-Champaign. Dept. of Computer Science). Department of Computer Science, University of Illinois at Urbana-Champaign, 1980.

[16] Brian Neil Mogensen. Goal-oriented conceptual clustering : The classifying attribute approach. *Coordinated Science Laboratory Report no. UILU-ENG-87-2257*, 1987.

[17] Jaideep Motwani, Ram Subramanian, and Pradeep Gopalakrishna. Critical factors for successful erp implementation : Exploratory findings from four case studies. *Comput. Ind.*, 56(6) :529–544, August 2005.

[18] Abdelkader Ouali, Samir Loudni, Yahia Lebbah, Patrice Boizumault, Albrecht Zimmermann, and Lakhdar Loukil. Efficiently finding conceptual clustering models with integer linear programming. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 647–654, 2016.

[19] Ruggero G. Pensa, Céline Robardet, and Jean-François Boulicaut. *A Bi-clustering Framework for Categorical Data*, pages 643–650. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[20] Mike Perkowitz and Oren Etzioni. Towards adaptive web sites : Conceptual framework and case study. *Artificial Intelligence*, 118(1) :245 – 275, 2000.

[21] 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, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 204–212, 2008.

[22] Lionel Robert, Ashley R. Davis, and Alexander McLeod. Erp configuration : Does situation awareness impact team performance ? *2011 44th Hawaii International Conference on System Sciences (HICSS 2011)*, 00(undefined) :1–8, 2011.

[23] Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. *An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases*, pages 16–31. Springer Berlin Heidelberg, 2004.

[24] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 577–584, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

Contraintes de Classement

Christian Bessiere[1] **Emmanuel Hebrard**[2] **George Katsirelos**[3] **Zeynep Kiziltan**[4] **Toby Walsh**[5]

[1] LIRMM, CNRS, Université de Montpellier [2] LAAS-CNRS, Université de Toulouse
[3] MIAT, INRA [4] Université de Bologne
[5] Université de Nouvelle-Galles du Sud

bessiere@lirmm.fr hebrard@laas.fr george.katsirelos@toulouse.inra.fr zeynep.kiziltan@unibo.it tw@cse.unsw.edu.au

Résumé

La notion de classement se manifeste dans de nombreux domaines : la recherche d'information, la planification de tournois, la bibliométrie, ou encore l'analyse statistique. Nous proposons une contrainte pour modéliser ce concept, ainsi que deux décompositions et des algorithmes de filtrage efficaces. Les résultats expérimentaux sur la minimisation de la corrélation entre classements démontrent l'intérêt de l'approche choisie. Ce papier est un résumé d'un article présenté à IJCAI-16 [1].

1 Introduction

Supposons que nous voulions déterminer le classement de joueurs de tennis sur un ensemble de tournois. La méthode de Kemeny-Young est la seule méthode à la fois *neutre*, *cohérente* et *Condorcet*. Elle calcule le classement qui minimise la somme des distances de *Kendall tau*, i.e., du nombre de divergences, pour chaque paire de joueurs, avec le classement d'un tournoi. Déterminer ce classement est NP-complet pour 4 tournois, et le calculer par résolution d'un CSP nécessite de représenter la notion de classement.

Un classement peut avoir des ex æquo, au contraire d'une permutation. Par exemple, 12225 et 12345 sont des classements mais seul le deuxième est une permutation. Il existe une contrainte globale élégante et efficace pour la notion de permutation, pour laquelle Régin a proposé un algorithme établissant la cohérence d'arc en $O(n^2d^2)$ [5]. Pourtant, aucun outil de résolution ne propose d'algorithme pour la contrainte de classement. Nous comblons ce manque dans ce papier.

2 La contrainte de classement

Définition 1 *Une séquence R est un* classement *si et seulement si $R = (1)$, ou $R = (x_1, \ldots, x_{m+1})$ avec $x_{m+1} = x_m$ ou $x_{m+1} = m + 1$ et $(x_1, \ldots, x_m)$ est un classement. La contrainte* $\textsc{Classement}(X_1, \ldots, X_n)$ *est satisfaite si et seulement si il existe une permutation π telle que $(X_{\pi(1)}, \ldots, X_{\pi(n)})$ est un classement.*

Par exemple, $\textsc{Classement}([4, 1, 2, 2])$ est satisfaite, mais $\textsc{Classement}([3, 1, 4, 3])$ ne l'est pas.

Il est possible d'exprimer cette contrainte à l'aide de la contrainte $\textsc{Trié}(\mathbf{X}, \mathbf{Y})$ [3], qui impose que $\mathbf{Y} = Y_1, \ldots, Y_n$ soit une permutation croissante de la séquence $\mathbf{X} = X_1, \ldots, X_n$:

$$\begin{aligned} &\textsc{Trié}(\mathbf{X}, \mathbf{Y}), \quad Y_1 = 1 \\ &\quad Y_i = Y_{i-1} \vee Y_i = i \quad \forall i \in [2, n] \end{aligned}$$

Un second modèle utilise la contrainte $\textsc{Cgc}(\mathbf{X}, V, \mathbf{Y})$ [4], où $\mathbf{Y} = \{Y_v \mid v \in V\}$, qui impose à chaque valeur $v \in V$ un nombre d'occurrences dans $\mathbf{X}$ égal à Y_v.

$$\begin{aligned} \textsc{Cgc}(\mathbf{X}, \{1, \ldots, n\}, \mathbf{Y}), \quad Y_1 &= Z_1 \\ Z_i &= Z_{i-1} + Y_i \quad \forall i \in [2, n] \\ Z_i &\geq i \quad \forall i \in [1, n] \\ Y_i = 0 \iff Z_{i-1} &\geq i \quad \forall i \in [2, n] \end{aligned}$$

Établir la cohérence d'arc sur l'une ou l'autre de ces décompositions n'est pas suffisant pour établir la cohérence d'arc (de bornes) sur la contrainte Classement.

Arc cohérence. Le problème de l'existence d'un support peut être reformulé comme un problème de flot maximum lexicographique [2] (e.g., avec des coûts exponentiels). Un flot dans ce réseau est une séquence R qui respecte une condition proche de celle d'un classement : l'égalité $x_{m+1} = m + 1$ dans la définition 1 est remplacée par $x_{m+1} \geq m+1$. Si le flot maximum n'est

pas un classement, il existe un indice i dans la séquence tel que le préfixe de taille $i-1$ (i.e., $x_1, \ldots, x_{i-1}$, les $i-1$ plus petites valeurs) est un classement, et $x_i > i$. Or, la séquence $R = x_1, \ldots, x_n$ donnée par un flot maximum est maximale dans l'ordre lexicographique. Il s'en suit qu'il n'existe pas de classement préfixe dont le maximum est supérieur ou égal à x_i et de longueur $\leq i$. Cette contrainte peut être rajoutée au réseau en changeant la capacité d'un seul arc. Le nombre de révisions, et donc d'itérations est borné par $O(n^2)$.

Cohérence de bornes. L'algorithme glouton suivant calcule un support pour la cohérence de bornes en temps $O(n \log n)$. Il maintient une partition des variables entre assignées (A) et non-assignées (U) :

SupportCB : A chaque itération, si aucune variable dans U ne contient la valeur $|A|+1$, alors un échec est renvoyé. Sinon, la variable X_i avec la plus petite borne supérieure est choisie parmi celles-ci. Lors d'une itération, les trois étapes suivantes ont lieu :

- X_i est affectée à $|A|+1$ et déplacée de U vers A.
- Pour un ensemble F initialement vide, et tant qu'il existe des variables dans U dont la borne supérieure est inférieure à $|A|+|F|+1$ ces variables sont déplacées de U vers F.
- Toutes les variables dans F prennent la valeur $|A|$ et sont déplacées de F vers A. Un échec est renvoyé si ce n'est pas possible.

Théorème 1 *L'algorithme SupportCB retourne un support de bornes s'il en existe un, et renvoie un échec sinon, en temps* $O(n \log n)$.

Il existe trois raisons pour l'incohérence de bornes :

Définition 2 (Valeur saturée/échec) *Une valeur v est saturée s'il existe exactement v variables dont le domaine a une intersection non nulle avec l'intervalle $[1, v]$. De plus, si ce nombre est strictement inférieur à v, alors v est une valeur échec.*

S'il existe une valeur saturée v, alors le domaine de toute variable contenant v peut être restreint à son intersection avec $[1, v]$. S'il existe une valeur échec, alors la contrainte est incohérente.

Définition 3 ((super-)intervalles de Hall) *Soit $V_{\sqsubseteq}(a,b) = \{X \mid D(X) \subseteq [a,b]\}$ et $S(a,b) = |V_{\sqsubseteq}(a,b)|$. $[a,b]$ est un intervalle de Hall si $S(a,b) = b-a+1$, et un super-intervalle de Hall si $S(a,b) > b-a+1$.*

S'il existe un super-intervalle de Hall $[a,b]$, alors aucune variable hors de $V_{\sqsubseteq}(a,b)$ ne peut prendre de valeur dans l'intervalle $[b+1, a+S(a,b)-1]$.

Enfin, une affectation peut être incohérente parce qu'elle étendrait un ensemble de (super-)intervalles jusqu'à vider le domaine d'une autre variable. Par exemple, une variable $X_j \notin V_{\sqsubseteq}(a,b)$ participerait à cet intervalle si une valeur dans $[a,b]$ lui été affectée, avec pour conséquence l'extension des valeurs interdites $[b+1, a+S(a,b)-1]$, jusqu'à possiblement recouvrir le domaine d'une variable X_j.

3 Résultats expérimentaux

Nous avons évalué les différents algorithmes et décompositions sur le problème de la minimisation de la corrélation entre deux classements. La figure 1 montre, pour chaque taille de séquence dans $[5, 20]$, le temps CPU moyen et le ratio d'instances non résolues en moins de 30 minutes (en pointillé) sur des domaines générés de façon aléatoire et pour quatre méthodes : les deux décompositions de la section 2 (Trié, Cgc) ; un algorithme complet pour la cohérence de bornes "force brute" (singleton) ; et un algorithme incomplet basé sur les règles de filtrage de la section 2 (filtrage).

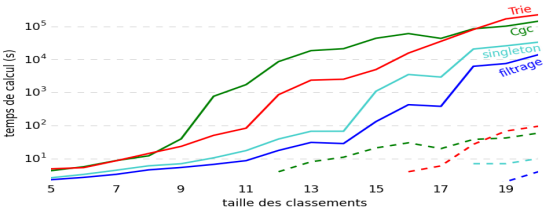


FIGURE 1 – Non-corrélation : Intervalles aléatoires

4 Conclusion

Nous avons proposé une contrainte globale de classement. Établir la cohérence d'arc pour cette contrainte est polynomial, mais coûteux. Nous avons donc proposé un algorithme pour la cohérence de bornes permettant un gain en performance d'environ un ordre de grandeur par rapport à une décomposition.

Références

[1] C. Bessiere, E. Hebrard, G. Katsirelos, Z. Kiziltan, T. Walsh. Ranking constraints. *IJCAI*, p. 705–711, 2016.

[2] D. Kozen. Lexicographic flow. Technical Report, Cornell University, 2009.

[3] W.J. Older, G.M. Swinkels, M.H. van Emden. Getting to the real problem : experience with BNR Prolog in OR. *PAP*, p. 465-478, 1995.

[4] A. Oplobedu, J. Marcovitch, Y. Tourbier. CHARME : Un langage industriel de programmation par contraintes, illustré par une application chez Renault. *Workshop on Expert Systems and their Applications*, p. 55-70, 1989.

[5] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. *AAAI*, p. 362-367, 1994.

Décomposition de contraintes basée sur les propriétés traitables

Achref El Mouelhi
Marseille - France
elmouelhi.achref@gmail.com

Résumé

Des efforts considérables ont été consacrés à l'identification des propriétés traitables pour les problèmes de satisfaction de contraintes (CSP) lors de ces deux dernières décennies. Récemment, certains travaux ont montré l'intérêt de ces propriétés traitables d'un point de vue différent. Par exemple, il est connu que s'il n'existe aucun triangle cassé sur tous les couples de valeurs d'une variable donnée dans une instance de CSP binaire arc-cohérente, cette variable peut être éliminée tout en préservant la satisfiabilité. Ensuite, il a été prouvé que, même si cette règle ne peut être appliquée à cause de la présence d'un triangle cassé, il se peut qu'il existe un couple de valeurs sur lesquelles aucun triangle cassé ne se produit. Si c'est le cas, on peut effectuer une opération de réduction de domaine qui consiste à fusionner ces valeurs tout en préservant la satisfiabilité.

Dans ce papier, nous montrons que la présence de propriétés traitables dans des contraintes non-binaires suffit pour qu'elles soient décomposables en un ensemble de contraintes d'arité inférieure tout en préservant l'équivalence. Plus précisément, nous prouvons que la présence de la propriété bijection ou l'absence des triangles cassés duaux dans une contrainte non-binaire permet de la décomposer tout en préservant l'équivalence.

Ceci est un résumé d'un article publié en langue anglaise à ICTAI'16 [6].

1 Introduction

Les *problèmes de satisfaction de contraintes* (CSP, pour Constraint Satisfaction Problems en anglais) se situent au centre de nombreuses applications en intelligence artificielle. Formellement, une instance CSP est un triplet $I = (X, D, C)$, où $X = \{x_1, \ldots, x_n\}$ est un ensemble de n **variables**, chaque variable x_i possède un ensemble fini de valeurs $D(x_i)$, appelé domaine. C est un ensemble fini de **contraintes**. Chaque contrainte c_i porte sur un ensemble de variables, noté $S(c_i) = \{x_{i_1}, ..., x_{i_{a_i}}\} \subseteq X$ et appelé la **portée** de la contrainte et autorise un ensemble de combinaisons de valeurs, noté $R(c_i) \subseteq D(x_{i_1}) \times ... \times D(x_{i_{a_i}})$ et appelé la **relation** de c_i. $\mid S(c_i) \mid$ appelé arité de c_i, correspond au nombre de variables figurant dans la portée de c_i. Une instance CSP est dite *binaire* si l'arité de chaque contrainte est égale à deux (dans ce cas, on note par $c_{i,j}$ la contrainte avec $S(c_{i,j}) = \{x_i, x_j\}$), sinon elle est dite *non-binaire* (ou *d'arité quelconque*). Une instance CSP est dite *ternaire* lorsque l'arité de chaque contrainte est inférieure ou égale à trois. Étant donnés une contrainte c_ℓ, un tuple $t_\ell \in R(c_\ell)$ et x_I un sous-ensemble de variables dans $S(c_\ell)$, $t_\ell[x_I]$, $c_\ell[x_I]$, $S(c_\ell)[x_I]$ et $R(c_\ell)[x_I]$ désignent respectivement la restriction de t_ℓ, c_ℓ, $S(c_\ell)$ et $R(c_\ell)$ aux variables de x_I.

2 Décomposition basée sur la présence de la propriété bijection

Une relation $R(c_{i,j})$, associée à une contrainte binaire $c_{i,j}$, est une **bijection** [1] si chaque valeur dans $D(x_i)$ est compatible avec une seule valeur dans $D(x_j)$ (et inversement). Nous disons aussi que $c_{i,j}$ satisfait la propriété bijection.

Le premier résultat montre que l'existence d'une sous-contrainte binaire qui satisfait la propriété bijection dans une contrainte d'arité quelconque permet de la décomposer en deux contraintes d'arité inférieure tout en préservant l'équivalence.

Théorème 1 *Étant donnée une contrainte d'arité quelconque c_ℓ, s'il y a deux variables $x_i, x_j \in S(c_\ell)$ telles que $R(c_\ell)[\{x_i, x_j\}]$ est une bijection, alors c_ℓ peut être décomposée en deux contraintes $c_{i,j}$ et c'_ℓ avec $S(c'_\ell) = S(c_\ell) \setminus \{x_i\}$ (ou $c_{i,j}$ et c''_ℓ avec $S(c''_\ell) = S(c_\ell) \setminus \{x_j\}$) tout en préservant l'équivalence.*

Exemple 1 *Le tableau ci-dessous illustre le cas d'une contrainte décomposable c_ℓ puisque $R(c_\ell)[\{x_i, x_j\}]$ est une bijection. Le second montre les deux contraintes obtenues après décomposition de la première.*

$R(c_\ell)$			
x_i	x_j	x_m	x_k
a	b	c	d
a'	b'	c'	d
a''	b''	c''	d'

$R(c_{i,j})$	$R(c'_\ell)$
x_i x_j	x_i x_m x_k
a b	a c d
a' b'	a' c' d
a'' b''	a'' c'' d'

Nous généralisons la propriété bijection aux CSP d'arité quelconque. Avant cela, nous devons préciser qu'une première tentative a eu lieu dans [4] sans qu'elle représente pour autant une généralisation de la propriété. Mais elle peut être considérée comme une simple extension.

Définition 1 (G-bijection) *Étant donnée une contrainte d'arité quelconque c_ℓ, la relation $R(c_\ell)$ satisfait **G-bijection** (pour Generalalized bijection) s'il existe deux sous-ensembles disjoints x_I et x_J avec $S(c_\ell) \supseteq x_I \cup x_J$ tels que $\forall v_I \in R(c_\ell)[x_I]$, il existe un unique $v_J \in R(c_\ell)[x_J]$ tel que $(v_I, v_J) \in R(c_\ell)$.*

Il est claire que si $|x_I| = |x_J| = 1$, alors G-bijection correspond exactement à la propriété bijection définie pour les contraintes binaires. Aussi, si $S(c_\ell) = x_I \cup x_J$, nous parlons d'une présence totale de G-bijection dans la contrainte c_ℓ, autrement la présence est partielle.

Le théorème précédent peut évidemment être généralisé s'il existe deux sous-ensembles de variables x_I et x_J (avec $x_I \cup x_J \subseteq S(c_\ell)$) qui satisfont la G-bijection (présence partielle). Également, s'il s'agit d'une présence complète de G-bijection dans c_ℓ et si $\mid x_I \mid$ ou $\mid x_J \mid$ est égale à 1, alors c_ℓ est décomposable en un arbre de contraintes binaires équivalentes. c_ℓ est décomposable en une chaîne de contrainte binaire si pour tout x_I, x_J et x_K tels que $S(c_\ell) = x_I \cup x_J \cup x_K$, $c_\ell[x_I \cup x_J]$ satisfait la G-bijection.

3 Décomposition basée sur l'absence des triangles cassés duaux

DBTP (pour **Dual Broken-Triangle Property**) [5] est la première extension de la propriété traitable, BTP, aux CSP d'arité quelconque. En effet, il a été démontré que l'absence des triangles cassés duaux (DBT) sur la dernière contraint de chaque triplet de contraintes d'une instance CSP d'arité quelconque munie d'un ordre $\prec$ sur les variables permet de déduire qu'elle est traitable.

Dans ce papier, nous montrons que l'absence des triangles cassés duaux dans le primal [2] prouve que la contrainte est décomposable en une clique de contraintes binaires équivalentes.

Notation 1 *Étant donnée une contrainte c_ℓ d'arité quelconque, nous notons par $Inst(c_\ell) = (X', D', C')$ une sous-instance de CSP binaire avec :*

- $X' = \{x_i \in S(c_\ell)\}$
- $D' = \{D(x_i) \mid x_i \in X'\}$
- $C' = \{c_{i,j} \mid x_i, x_j \in S(c_\ell)$ *et* $R(c_{i,j}) = R(c_\ell)[\{x_i, x_j\}]\}$

Théorème 2 *Considérant une contrainte c_ℓ d'arité quelconque, si $Inst(c_\ell)$ ne contient aucun triangle cassé dual, alors c_ℓ peut être remplacée par les contraintes binaires de $Inst(c_\ell)$ tout en préservant l'équivalence.*

En pratique, nous n'avons pas à convertir c_ℓ en $Inst(c_\ell)$ pour vérifier si un tuple $t \in R(c_{i,j})$. Nous pouvons utiliser directement $c_\ell[\{x_i, x_j\}]$.

4 Conclusion

Après l'utilisation des propriétés traitables dans la réduction de la taille de l'espace de recherche par l'élimination de variables et la fusion de valeurs, nous avons montré dans ce travail comment ces propriétés peuvent être utilisées pour décomposer les contraintes d'arité quelconque en contraintes d'arité inférieure tout en préservant l'équivalence. Nous avons aussi établi le lien entre ces règles de décomposition et GMvD (General Multivalued Dependency) et GID (General Interdependency) [3]. Il serait intéressant de vérifier si les propriétés traitables peuvent être notamment utilisées pour la compression des contraintes tables ou pour définir des nouvelles contraintes globales.

Références

[1] Philippe David. When functional and bijective constraints make a CSP polynomial. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 224–231, 1993.

[2] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34 :1–38, 1987.

[3] Achref El Mouelhi. On the decomposition of non-binary constraint into equivalent binary constraints. In *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI, 2015*, pages 9–16, 2015.

[4] 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, 11-12 July 2013*, 2013.

[5] Achref El Mouelhi, Philippe Jégou, and Cyril Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. *Constraints*, 20(4) :383–413, 2015.

[6] Achref El Mouelhi. Constraint decomposition based on tractable properties. In *28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI, 2016*, pages 226–230, 2016.

Un nouveau VRPTW statique et stochastique : vers une modélisation en deux étapes plus réaliste

Michael Saint-Guillain[1,2] * **Christine Solnon**[2] **Yves Deville**[1]

[1] ICTEAM, Université catholique de Louvain, Belgique
[2] Université de Lyon, CNRS
[2] INSA-Lyon, LIRIS, UMR5205, F-69621, France
{michael.saint,yves.deville}@uclouvain.be christine.solnon@insa-lyon.fr

Résumé

Ce papier résume l'article [3] accepté à la conférence "EvoStar2017". Nous introduisons le SS-VRPTW-CR, un nouveau problème de tournées de véhicules statique et stochastique (SS-VRP) en deux étapes (*two-stage*). Lorsque les clients d'un VRPTW ne sont présents qu'avec une certaine probabilité, le moment auquel cette information est révélée est généralement considéré comme connu (par exemple, lorsque le véhicule quitte le client précédent). Le SS-VRPTW-CR permet de modéliser l'incertitude au niveau de ce moment même, où la présence (ou absence) du client est révélée.

1 Introduction

Nous introduisons un nouveau problème dans la famille des VRP stochastiques : *Static and Stochastic VRPTW with both random Customers and Reveal Times (SS-VRPTW-CR)*. En plus des fenêtres temporelles (TW), ce problème stochastique a la particularité de ne faire aucune hypothèse sur le moment auquel une requête potentielle apparaît (si celle-ci apparaît). Cette information, appelée *customer reveal time* (R), ainsi que la présence de chaque client (C) est considérée comme stochastique. Cela contraste avec les études existantes traitant du SS-VRPTW, qui considèrent que l'information de présence (ou d'absence) d'un client est révélée à un moment prédéfini, par exemple lorsque le véhicule quitte le client précédent.

Dans une récente étude [1], les auteurs font remarquer qu'avec l'augmentation de l'utilisation de technologies de l'information et de la communication, la demande d'un client (et par extension sa présence) est

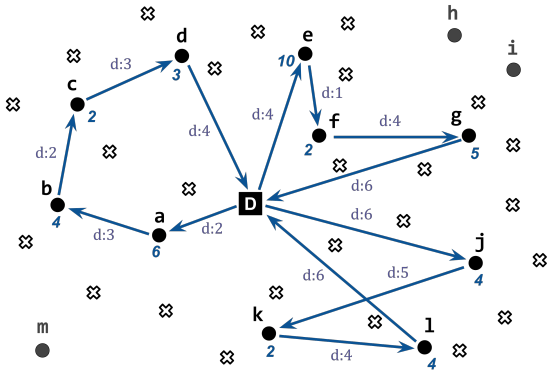


FIGURE 1 – Une solution a priori au SS-VRPTW-CR. Les points représentent les positions où les véhicules peuvent attendre. Les croix représentent les positions où les requêtes peuvent potentiellement apparaître.

à l'heure actuelle susceptible d'être révélée en temps réel. Dans ce contexte, l'ordre chronologique dans lequel les informations sont révélées ne dépend plus de la séquence de clients visités par les véhicules. En particulier, il est décrit comme paradoxal le fait que les études existantes supposent que la présence des clients soit révélée entièrement au début des opérations.

2 Données du problème et solutions

Une instance du SS-VRPTW se compose d'un horizon de temps H (discret), d'un dépôt, d'une flotte de véhicules et de deux ensembles de positions : les requêtes potentielles (C) et les points d'attentes (W). À chaque position dans C sont associées un ensemble

*Papier doctorant : Michael Saint-Guillain[1,2] est auteur principal.

de H requêtes potentielles, une pour chaque unité de temps de l'horizon où une requête est susceptible d'apparaître. On connaît alors la probabilité d'apparition de chaque requête potentielle, ainsi que les attributs de la requête le cas échéant, tels que sa TW, sa demande, son temps de service, *etc.*

Pour les requêtes ayant des TW, il est possible que certaines requêtes ne puissent être satisfaites. L'objectif est de calculer une solution a priori (avant le début des opérations) qui soit adaptée en temps réel (au moment où les demandes sont révélées) de sorte que l'espérance du nombre de requêtes rejetées soit minimale. Une solution a priori décrit, pour chaque véhicule, une séquence de positions où celui-ci doit attendre des requêtes potentielles environnantes pendant un temps donné. La procédure d'adaptation (*recourse strategy*) utilisée en temps réel consiste à 1) accepter ou non chaque requête qui vient d'être révélée, 2) servir les requêtes déjà acceptées. Déterminer si le véhicule associé à une requête peut la satisfaire est fait en temps polynomial, en fixant un ordre sur les requêtes potentielles associées à chaque position d'attente.

La Figure 1 montre un exemple de solution a priori. Dans cet exemple, l'un des véhicules passe par la position $a \in W$ qui devient alors, pendant 6 unités de temps, le point à partir duquel les requêtes environnantes pourront être satisfaites par le véhicule. Le véhicule partira ensuite pour le point d'attente $b \in W$, *etc.* La solution a priori (ou *first stage*) sert donc de patron à la construction de la solution finale (*second stage*). Plus la solution a priori est adaptée à la distribution de requêtes potentielles, plus la solution finale a de chance d'être en mesure de servir un grand nombre de requêtes parmi celles qui apparaîtront.

La Figure 2 montre ce à quoi peuvent ressembler les routes des véhicules lorsque, basées sur l'exemple de solution a priori de la Figure 1, certaines requêtes qui apparaissent durant les opérations sont satisfaites.

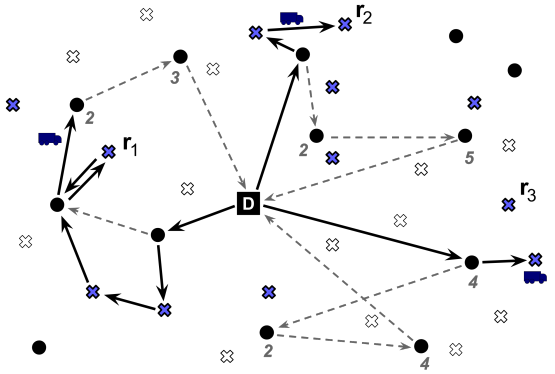


FIGURE 2 – La solution a priori (en pointillés) est adaptée afin de servir les requêtes qui apparaissent durant les opérations. Les croix pleines représentent les positions où une requête est déjà apparue, au moment de la journée considéré ici.

3 Applications et contributions

Le SS-VRPTW-CR s'applique naturellement à tout problème logistique où les clients, ou leurs requêtes, apparaissent de façon dynamique au cours des opérations ; à condition d'avoir accès à des données statistiques fiables et représentatives du comportement de chaque client, le SS-VRPTW-CR fournit un moyen d'appréhender le problème de façon anticipative, en particulier sous de fortes contraintes temporelles.

Dans [3], l'exemple d'un service d'assistance médicale à domicile (par exemple pour personnes âgées ou handicapées) est décrit. Dans ce contexte, un abonné au service demandant une assistance est garanti d'être servi dans un délai très court. En pratique, une requête impossible à satisfaire sera alors confiée à un service externe, générant un coût important.

Ce VRP stochastique peut être vu comme la version en deux étapes (*two-stage*) de la variante dynamique, dite multi-étape (*multistage*) : le Dynamic and Stochastic (DS-)VRPTW [2]. La contribution du SS-VRPTW-CR au DS-VRPTW est de fournir une borne supérieure à l'espérance du coût d'une solution sous ré-optimisation parfaite.

En effet, nous montrons comment il est possible, en définissant une stratégie dictant aux véhicules comment réagir aux différent évènements aléatoires, de calculer efficacement et de façon exacte l'espérance du coût d'une solution, pourtant soumise à un nombre exponentiel de scénarios probables. Un algorithme heuristique basé sur la recherche locale est proposé afin de trouver des solutions de qualité en un temps raisonnable, donnant un aperçu des résultats obtenus sur des instances générées aléatoirement.

Références

[1] Michel Gendreau, Ola Jabali, Walter Rei, Michel Gendreau, and Ola Jabali. Future Research Directions in Stochastic Vehicle Routing. 1655(October) :1–11, 2016.

[2] Michael Saint-Guillain, Yves Deville, and Christine Solnon. A Multistage Stochastic Programming Approach to the Dynamic and Stochastic VRPTW. In *CPAIOR*, pages 357–374, 2015.

[3] Michael Saint-Guillain, Christine Solnon, and Yves Deville. The Static and Stochastic Vehicle Routing Problem with both random Customers and Reveal Times. In *EvoStar*, page (to be published), 2017.

The Static and Stochastic VRP with Time Windows and both random Customers and Reveal Times

Michael Saint-Guillain[1,2], Christine Solnon[2], Yves Deville[1]

[1]ICTEAM, Université catholique de Louvain, Belgium
[2]LIRIS, Institut National des Sciences Appliquées de Lyon, France

Abstract. Static and stochastic vehicle routing problems (SS-VRP) aim at modeling and solving real life problems by considering uncertainty on the data. In particular, customer data may not be known with certainty. Before the beginning of the day, probability distributions on customer data are used to compute a first-stage solution that optimizes an expected cost. Customer data are revealed online, while the solution is executed, and a recourse strategy is applied on the first-stage solution to quickly adapt it. Existing SS-VRP variants usually make a strong assumption on the time at which a stochastic customer reveals its data (*e.g.*, when a vehicle arrives at the corresponding location). We introduce a new SS-VRP where customer reveal times are stochastic. We define first-stage solutions and a recourse strategy for this new problem. A key point is to introduce waiting locations that are used in the first stage-solution to wait for the realization of customer stochastic data. We show how to compute the expected cost of a first-stage solution in pseudo polynomial time, in the particular case where the vehicles are not constrained by a maximal capacity. We also introduce a local search-based approach for optimizing the first-stage solution, and introduce a *scale* parameter to tune the precision and cost of the expected cost computation. Experimental results on small to large instances demonstrate its efficiency and flexibility.

1 Introduction

The Vehicle Routing Problem (VRP) aims at modeling and solving a real life common operational problem, in which a set of customers must be visited using a fleet of vehicles. Each customer comes with a certain demand. In the VRP with Time Windows (VRPTW), each customer must be visited within a given time window. A feasible solution of the VRPTW is a set of vehicle routes, such that every customer is visited exactly once during its time window and that sum of the demands along each route does not exceed the corresponding vehicle's capacity. The objective is then to find an optimal feasible solution, where optimality is usually defined in terms of travel distances.

The classical deterministic VRP(TW) assumes that customer data are known with certainty before the computation of the solution. Contrary to standard academic formulations, real world applications usually have missing part of the problem data when computing a solution. For instance, only a subset of the customer demands may be known before online execution. Missing demands hence arrive in a dynamic fashion, while vehicles are on their route. In such a context, a solution should contain operational decisions that deal with current known demands, but should also anticipate potential unknown demands. Albeit uncertainty may be considered for various attributes of the VRP

(*e.g.*, travel times), we focus on situations where the customer data are unknown a priori, and we assume that we have some probabilistic knowledge on missing data (*e.g.*, probability distributions computed from historical data). This probabilistic knowledge is used to compute a first-stage solution which is adapted online when random variables are realized. Two different kinds of adaptations may be considered: *Dynamic and Stochastic VRPTW* (DS-VRPTW) and *Static and Stochastic VRPTW* (SS-VRPTW).

In the DS-VRPTW, the solution is re-optimized at each time-step, and this re-optimization involves solving an $\mathcal{NP}$-hard problem so that it is usually approximated with meta-heuristics as proposed, for example, in [1,2,3]. Note that the DS-VRPTW assumes a probabilistic knowledge on the potential requests. In contrary, in [4] for instance no prior knowledge is provided on the potential requests, which are then assumed to be uniformly distributed in the Euclidean plan.

In the SS-VRP(TW), no expensive reoptimization is allowed during online execution. When unknown information is revealed, the first stage solution is adapted online by applying a predefined recourse strategy whose time complexity is polynomial. In this case, the goal is to find a first stage solution that minimizes its total cost plus the *expected* extra cost caused by the recourse strategy. For example, in [5], the first stage solution is a set of vehicle tours which is computed offline with respect to probability distributions of customer demands. Real customer demands are revealed online, and two different recourse strategies are proposed: in the first one, each demand is assumed to be known when the vehicle arrives at the customer place, and if it is larger than or equal to the remaining capacity of the vehicle, then the first stage solution is adapted by adding a round trip to the depot to unload the vehicle; in the second recourse strategy, each demand is assumed to be known when leaving the previous customer and the recourse strategy is refined so that customers with null demands are skipped.

In this paper, we focus on the SS-VRPTW, and introduce a new variant where no strong assumption is made on the moment at which customer requests are revealed during the operations (contrary to most existing work that assume that customer requests are known either when arriving at the customer place, or when leaving the previous customer). In this new variant, called the *SS-VRPTW with random Customers and Reveal time* (SS-VRPTW-CR), the reveal times of customer requests are random variables. To handle uncertainty on reveal times, we introduce waiting locations when computing first-stage solutions: the routes computed offline visit waiting locations and a waiting time is associated with each waiting location. When a customer request is revealed, it is either accepted (if it is possible to serve it) or rejected. The recourse strategy then adapts routes so that all accepted requests are guaranteed to eventually be served. The goal is to compute the first-stage solution that minimizes the expected number of rejected requests.

Our motivating application is an on-demand health care service for elderly or disabled people. Health care services are provided directly at home by mobile medical units. Every person who's registered to the service can request a health care support at any moment of the day with the guarantee to be satisfied within a given time window. From historical data, we know, for each customer region and each time unit, the probability that a request appears. Given this stochastic knowledge, we compute a first-stage solution. When a request appears (online), the recourse strategy is used to decide

whether the request is accepted or rejected and to adapt medical unit routes. When a request is rejected, the system must rely on an external service provider in order to satisfy it. Therefore, the goal is to minimize the expected number of rejected requests.

Organization. In section 2, we review the existing studies on VRPs that imply stochastic customers. Section 3 formally defines the general SS-VRPTW-CR. Section 4 describes a recourse strategy for this problem. Section 5 shows how the expected number of rejected requests can be efficiently computed from a first stage solution and for a specific recourse strategy. Section 6 describes a local search-based approach for approximating an optimal first stage solution. Experimental results are analysed in section 7. Further research directions are finally discussed in section 8.

2 Related work

The most studied cases in SS-VRPs are stochastic customers (presence of customers are random variables), stochastic demands (quantities required by customers are random variables), and stochastic times (travel and/or service times are random variables). Since the SS-VRPTW-CR belongs to the first case, we focus this review on customers uncertainty.

The Traveling Salesman Problem (TSP) is a special case of the VRP with only one uncapacitated vehicle. [6] introduced the TSP with stochastic Customers (SS-TSP-C), and provided mathematical formulations and a number of properties and bounds. In particular, he showed that an optimal solution for the deterministic problem can be arbitrarily bad in case of uncertainty. [7] developed the first exact solution method for the SS-TSP-C, using the integer L-shaped method [8] to solve instances up to 50 customers. Heuristics for the SS-TSP-C have then been proposed (e.g. [9,10,11]) as well as meta-heuristics such as simulated annealing [12] or ant colony optimization [13].

The first SS-VRP with stochastic Customers (SS-VRP-C) has been studied by [9] as a generalization of the SS-TSP-C. [14] compared different heuristics. [5] considered a VRP with stochastic Customers and Demands (SS-VRP-CD). A customer demand is assumed to be revealed either when the vehicle leaves the previous customer or when it arrives at the customer's own location. Two different recourse strategies are proposed. For both strategies, closed-form mathematical expressions are provided to compute the expected total distance, provided a first stage solution. [15] and [16] developed the first exact algorithm for solving the SS-VRP-CD for instances up to 70 customers, by means of an integer L-shaped method. [17] later proposed a tabu search to efficiently approximate the solution. Experimentations are reported on instances with up to 46 customers.[18] later developed an adaptive memory programming metaheuristic for the SS-VRP-C and assess it on benchmarks with up to 483 customers and 38 vehicles.

Particularly close to the SS-VRPTW-CR is the SS-TSP-C with Deadlines [19]. Unlike the SS-VRPTW-CR, the set of customers is revealed at the beginning of the operations. A recent literature review on SS-TSP-C may be found in [20].

[21] considered a variant of the SS-VRPTW-C, the Courier Delivery Problem with Uncertainty. Unlike the SS-VRPTW-CR, authors assume that customer presences are not revealed at some random moment during the operations, but all at once at the beginning of the day (that is, after computing the first stage solution).

3 Description of the SS-VRPTW-CR

Input data. We consider a complete directed graph $G = (V, A)$ and a discrete time horizon $H = [1, h]$, where interval $[a, b]$ denotes the set of all integer values i such that $a \leq i \leq b$. To every arc $(i, j) \in A$ is associated a travel time (or distance) $d_{i,j} \in \mathbb{N}$. The set of vertices $V = \{0\} \cup W \cup C$ is composed of a depot 0, a set of m waiting locations $W = [1, m]$ and a set of n customer regions $C = [m + 1, m + n]$. We note $W_0 = W \cup \{0\}$ and $C_0 = C \cup \{0\}$. The fleet is composed of K uncapacitated vehicles.

We consider the set $R = C \times H$ of potential customer requests such that an element $(c, \Gamma) \in R$ represents a potential request revealed at time $\Gamma \in H$ for customer region c. To each potential request $r = (c, \Gamma) \in R$ is associated a deterministic demand $q_r \in [1, Q]$, a deterministic service duration $s_r \in H$ and deterministic time window $[e_r, l_r]$ with $\Gamma \leq e_r \leq l_r \leq h$. We note p_r the probability that r appears on vertex c at time Γ, and assume independence between request probabilities. Although our formalism imposes $\Gamma \geq 1$ for all potential requests, in practice a request may be known with certainty that is, with probability 1.

To simplify notations, a request $r = (c, \Gamma)$ can be written in place of its own region c. For instance, the distance $d_{v,c}$ can also be intuitively written $d_{v,r}$. Furthermore, we use Γ_r to denote the reveal time of a request $r \in R$ and c_r for its customer region.

First stage solution. The first-stage solution is computed offline, before the beginning of the time horizon. It consists in a set of K vehicle routes visiting a subset of the m waiting vertices, together with time variables denoted τ indicating how long a vehicle should wait on each vertex. More specifically, we denote (x, τ) a first stage solution to the SS-VRPTW-CR, where:

- $x = \{x_1, ..., x_K\}$ defines a set of K sequences of waiting vertices of W, such that each sequence x_k starts and ends with 0 and each vertex of W occurs at most once in x. We note $W^x \subseteq W$ the set of waiting vertices visited in x.
- $\tau : W^x \to H$ associates a waiting time $\tau_w \geq 1$ to every waiting vertex $w \in W^x$;
- Each sequence $\langle 0, w_1, ..., w_{m'}, 0\rangle$ in x is such that the vehicle is back to the depot before the end of the day.

In other words, x defines a *Tour Orienteering Problem* (TOP, see [22]) to which each visited location is assigned a waiting time by τ. Given a first stage solution (x, τ), we define $on(w) = [\underline{on}(w), \overline{on}(w)]$ for each vertex $w \in W^x$ such that $\underline{on}(w)$ (resp. $\overline{on}(w)$) is the arrival (resp. departure) time on w. In a sequence $\langle 0, w_1, ..., w_{m'}, 0\rangle$ in x, we then have $\underline{on}(w_i) = \overline{on}(w_{i-1}) + d_{w_{i-1},w_i}$ and $\overline{on}(w_i) = \underline{on}(w_i) + \tau_{w_i}$ for $i \geq 1$ and assume both $\overline{on}(0) = 1$ and $\underline{on}(0) = \overline{on}(w_{m'}) + d_{w_{m'},0} \leq h$. Figure 1 (left) illustrates an example of first stage solution on a basic SS-VRPTW-CR instance.

Recourse strategy and second stage solution. A recourse strategy $\mathcal{R}$ states how a second stage solution is gradually constructed as requests are dynamically revealed. In this paragraph, we define the properties of a recourse strategy. An example of recourse strategy is given in Section 4.

Let $\xi \subseteq R$ be the set of requests that reveal to appear by the end of the horizon H. The set ξ is also called a *scenario*. We note $\xi^t \subseteq \xi$ the set of requests appearing at time

$t \in H$, *i.e.*, $\xi^t = \{r \in \xi : \Gamma_r = t\}$. We note $\xi^{1..t} = \xi^1 \cup ... \cup \xi^t$ the set of requests appeared up to time t.

A second stage solution is incrementally constructed at each time unit by following the skeleton provided by the first stage solution (x, τ). At a given time t of the horizon, we note (x^t, A^t) the current state of the second stage solution:

- $\diamond$ x^t defines a set of vertex sequences describing the route operations performed up to time t. Unlike x, we define x^t on a graph that also includes the customer regions. Operations described in x^t must satisfy the time window and vehicle capacity constraints imposed by the VRPTW.
- $\diamond$ $A^t \subseteq \xi^{1..t}$ is the set of *accepted* requests up to time t. Requests of $\xi^{1..t}$ that do not belong to A^t are said to be *rejected*.

We distinguish between requests that are accepted and those that are both accepted and satisfied. Up to a time t, an accepted request is said to be *satisfied* if it is visited in x^t by a vehicle. Accepted requests that are no yet satisfied must be *guaranteed to be eventually satisfied* according to their time window.

Figure 1(right) illustrates an example of second stage solution being partially constructed at some moment of the time horizon.

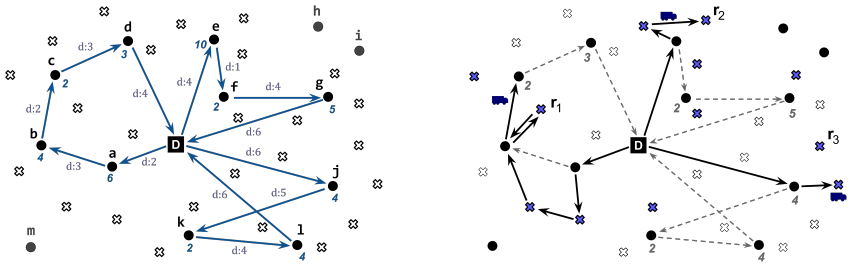


Fig. 1. On the left: first stage solution with $K = 3$ vehicles. The depot, waiting vertices and customer regions are represented by a square, circles and crosses respectively. Arrows represent vehicle routes and integers indicate waiting times at waiting locations. Values preceded by 'd' indicate travel times. Waiting vertices h, i and m are not part of the first stage solution. Here $\overline{on}(D) = 1$, $\underline{on}(a) = 3$, $\overline{on}(a) = 9$, $\underline{on}(b) = 12$, $\overline{on}(b) = 16$, etc.
On the right: partial second stage solution (plain arrows). Filled crosses are accepted requests. Some accepted requests, such as r_1, have been satisfied (or the vehicle is currently traveling towards the location, *e.g.*, r_2), while some others are not yet satisfied (*e.g.*, r_3).

Before starting the operations (time 0), x^0 is a set of K sequences that only contain vertex 0, and $A^0 = \emptyset$. At each time unit $t \in H$, given a first stage solution (x, τ), a previous state (x^{t-1}, A^{t-1}) of the second stage solution and a set ξ^t of requests appearing at time t, the new state (x^t, A^t) is obtained by applying a specific recourse strategy $\mathcal{R}$:

$$(x^t, A^t) = \mathcal{R}\big((x, \tau), (x^{t-1}, A^{t-1}), \xi^t\big). \tag{1}$$

A necessary property of a recourse strategy is to avoid reoptimization. We consider that $\mathcal{R}$ avoids reoptimization if the computation of (x^t, A^t) is achieved in polynomial time.

We note $\text{cost}(\mathcal{R}, x, \tau, \xi) = |\xi \setminus A^h|$ the final cost of a second stage solution with respect to a scenario ξ, given a first stage solution (x, τ) and under a recourse strategy $\mathcal{R}$. This cost is the number of requests that are rejected at the end h of the time horizon.

Optimal first stage solution. An optimal first stage solution (x, τ) to the SS-VRPTW-CR minimizes the expected cost of the second stage solution under a given strategy $\mathcal{R}$, satisfying statements (2)-(3):

$$\text{(SS-VRPTW-CR)} \quad \underset{x,\tau}{\text{Minimize}} \;\; \mathcal{Q}^{\mathcal{R}}(x, \tau) \tag{2}$$

$$\text{s.t.} \quad (x, \tau) \text{ is a first stage solution.} \tag{3}$$

The objective function $\mathcal{Q}^{\mathcal{R}}(x, \tau)$, which is nonlinear in general, determines the *expected number of rejected requests*, i.e. requests that fail to be visited under recourse strategy $\mathcal{R}$ and first stage solution (x, τ):

$$\mathcal{Q}^{\mathcal{R}}(x, \tau) = \sum_{\xi \subseteq R} \Pr(\xi)\, \text{cost}(\mathcal{R}, x, \tau, \xi) \tag{4}$$

where $\Pr(\xi)$ defines the probability of scenario ξ. Since we assume independence between requests, we have $\Pr(\xi) = \prod_{r \in \xi} p_r \cdot \prod_{r \in R \setminus \xi} (1 - p_r)$.

4 Description of a recourse strategy

In order to avoid reoptimization, the set R of potential requests is ordered. Furthermore, given a first stage solution (x, τ) that visits the set W^x of waiting locations, each potential request $r = (c, \Gamma) \in R$ is assigned to exactly one waiting vertex (and hence, a vehicle) in W^x.

Informally, the recourse strategy accepts a new request if it is possible for the vehicle associated to its corresponding waiting vertex location to adapt its first stage tour to visit the customer. The vehicle will then travel from the waiting location to the customer and return to the waiting location. Time window constraints should be respected, and the already accepted requests should not be perturbed. In the recourse strategy we propose here, we assume the vehicles not to be constrained by a maximal capacity.

Request ordering. Before computing first-stage solutions, we order R by increasing reveal time Γ_r first, end of time window l_r second and request number r to break ties. Let $<_R$ denote this total strict order on R. Whereas the remaining of the paper is based on the assumption of total order on Γ_r, the ordering criteria may be modified without loss of generality (*e.g.*, replacing l_r by e_r), as long as the total order remains strict and primarily based on Γ_r, i.e. $\forall r_1, r_2 \in R, \Gamma_{r_1} < \Gamma_{r_2} \Rightarrow r_1 <_R r_2$.

Request assignment according to a first stage solution. Given a first-stage solution (x, τ), we assign each request of R to a waiting vertex visited in (x, τ). This assignment is computed for each first stage solution (x, τ) before the application of the recourse strategy. As an optimally fair distribution of the potential requests might be excessively expensive to compute, we propose the following heuristic.

Let $t^{\min}_{r,w} = \max\{\underline{on}(w),\ \Gamma_r,\ e_r - d_{w,r}\}$ be the minimum time for leaving waiting location w to satisfy request r. Indeed, a vehicle cannot handle r before (1) the vehicle is on w, (2) r is revealed, and (3) the beginning e_r of the time window minus the time $d_{w,r}$ needed to go from w to r.

Let $t^{\max}_{r,w} = \min\{l_r - d_{w,r},\ \overline{on}(w) - d_{w,r} - s_r - d_{r,w}\}$ be the latest time at which a vehicle can handle r (which also involves a service time s_r) from waiting location w and still leave it in time $t \leq \overline{on}(w)$.

Given a first stage solution (x, τ), we assign each request $r \in R$ either to a waiting vertex of W^x or to $\perp$ (to denote that r is not assigned). We note $w(r)$ this assignment which is computed as follows:

- ◇ Let $W^x_r = \{w \in W^x : t^{\min}_{r,w} \leq t^{\max}_{r,w}\}$ be the set of feasible waiting locations for r
- ◇ If $W^x_r = \emptyset$ then set $w(r)$ to $\perp$ (r is always rejected)
- ◇ Else set $w(r)$ to the feasible vertex of W^x_r that has the least number of requests already assigned to it (break further ties w.r.t. vertex number)

Once finished, the request assignment ends up with a partition $\{\pi_\perp, \pi_1, ..., \pi_K\}$ of R, where π_k is the set of requests assigned to the waiting vertices visited by vehicle k and $\pi_\perp$ is the set of unassigned requests (such that $\mathrm{w}(r) = \perp$). We note $\pi^w_k \subseteq \pi_k$ the set of requests assigned to $w \in W^x$ in route k. We note $\mathrm{fst}(\pi^w_k)$ the first request of π^w_k according to order $<_R$, and for each request $r \in \pi^w_k$ such that $r \neq \mathrm{fst}(\pi^w_k)$ we note $\mathrm{prv}(r)$ the request of π^w_k that immediately precedes r according to order $<_R$.

Using the recourse strategy to adapt a first stage solution at a current time t. At each time step t, the recourse strategy is applied to compute the second stage solution (x^t, A^t), given the first stage solution (x, τ), the second stage solution (x^{t-1}, A^{t-1}) at the end of time $t-1$, and the incoming requests ξ^t.

A^t is the set of accepted requests. It is initialized with A^{t-1}. Then, each incoming request of ξ^t is considered (taken by increasing order of $<_R$) and either accepted (added to A^t) or rejected (not added to A^t) by applying the following decision rule:

- ◇ Let k be the vehicle associated with r (i.e., $r \in \pi^k$)
- ◇ Let $y : R \to H$ be the function returning the time at which k finishes to satisfy all accepted requests that precede r (according to $<_R$) and reaches waiting vertex $\mathrm{w}(r)$. Namely, $y(r)$ is the time at which k is available for r and is defined by:
  - ⋆ If $r = \mathrm{fst}(\pi^w_k)$, then $y(r) = \underline{on}(w)$
  - ⋆ else if $\mathrm{prv}(r) \notin A^t$ then $y(r) = y(\mathrm{prv}(r))$
  - ⋆ else $y(r) = \max(y(\mathrm{prv}(r)) + d_{w,\mathrm{prv}(r)}, e_{\mathrm{prv}(r)}) + s_{\mathrm{prv}(r)} + d_{\mathrm{prv}(r),w}$

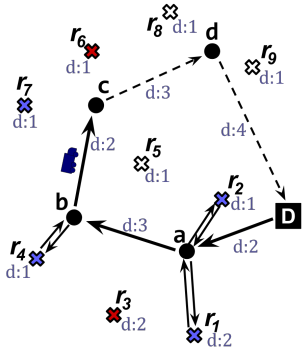
  If $y(r)$ allows k to reach r during its time window and to arrive in time to its next waiting location (*i.e.*, $y(r) \leq t^{\max}_{r,\mathrm{w}(r)}$) then r is accepted and added to A^t; otherwise it is rejected.

Once A^t has been computed, vehicle operations for time unit t must be decided. Vehicles operate independently of each other. If vehicle k is traveling between a waiting location and a customer region, or if it is serving a request, then its operation remains unchanged; Otherwise, let w be the current waiting location (or the depot) of vehicle k:

- ◇ If $t = \overline{on}(w)$, the operation for k is "travel from w to the next waiting vertex (or the depot), as defined in the first stage solution"

- ◇ Otherwise, let $P = \{r \in \pi_k^w | c_r \notin x^t \wedge (r \in A^t \vee t < \Gamma_r)\}$ be the set of requests of π_k^w that are not yet satisfied and that are either accepted or with unknown revelation
  - ⋆ If $P = \emptyset$, then the operation for k is "travel back to the depot"
  - ⋆ Otherwise, let r^{next} be the smallest element of P according to $<_R$
    - → If $t < t^{\min}_{r^{\text{next}},w}$, then the operation for k is "wait until $t+1$"
    - → Otherwise, the operation is "travel to r^{next}, serve it and come back to w"

Figure 2 shows an example of second stage solution at a current time $t = 17$, from an operational point of view.



r	w(r)	Γ_r	$[e_r, l_r]$	$t^{min}_{r,a}$	$t^{max}_{r,a}$	status	y(r)	sat(r)
a	Arrival time : 3			Departure time : 9				
1	a	2	2 , 6	3	4	satisf.	3	5
2	a	2	2 , 8	3	7	satisf.	7	8
3	a	5	5 , 15	5	5	rejected	9	n.a.
b	Arrival time : 12			Departure time : 16				
4	b	10	13, 16	13	13	satisf.	12	14
5	b	13	13 , 17	13	14	absent !	15	n.a.
c	Arrival time : 18			Departure time : 20				
6	c	16	16 , 20	18	18	accepted	18	19
7	c	17	17 , 19	18	18	rejected	20	n.a.
d	Arrival time : 23			Departure time : 26				
8	d	18	18 , 25	23	24	unknown	23	?
9	d	20	20 , 30	23	24	unknown	?	?

current time: t = 17

Fig. 2. Example of second stage solution at time $t = 17$, under strategy $\mathcal{R}1$. A filled cross represents a request that appeared, an empty one a request that is either still unknown (*e.g.*, r_8) or revealed as being absent (that is didn't appear, *e.g.*, r_5). Here $\pi_k = \langle r_a, r_1, \ldots, r_9 \rangle$ is the sequence of requests assigned to the vehicle, according to (x, τ). We assume $q_r = s_r = 0, \forall r \in R$. $sat(r)$ represents, for a request r, the time at which r gets satisfied.

5 Expected cost of second stage solutions

Provided a recourse strategy $\mathcal{R}$ and a first stage solution (x, τ) to the SS-VRPTW-CR, a naive approach for computing $\mathcal{Q}^{\mathcal{R}}(x, \tau)$ would be to literally follow equation (4), therefore using the strategy described by $\mathcal{R}$ in order to confront (x, τ) to each and every possible scenario $\xi \subseteq R$. Because there is an exponential number of scenarios with respect to $|R|$, this naive approach is not affordable in practice. In this section, we show how the expected number of rejected requests $\mathcal{Q}^{\mathcal{R}}(x, \tau)$ under the recourse strategy described in Section 4 may be computed in $\mathcal{O}(nh^2)$ using closed form expressions, in the special case where vehicles are of infinite capacity.

We assume that the potential request probabilities are independent from each other such that, for any couple of requests $r, r' \in R$, the probability $p_{r \cap r'}$ that both requests appear is given by $p_{r \cap r'} = p_r \cdot p_{r'}$.

Expected cost. $\mathcal{Q}^{\mathcal{R}}(x, \tau)$ is equal to the expected number of rejected requests, which in turn is equal to the expected number of requests that reveal to appear minus the

expected number of accepted requests. The expected number of revealed requests is given by the sum of all request probabilities, whereas the expected number of accepted requests is equal to the sum, for every request r, of the probability that it belongs to A^h:

$$\mathcal{Q}^{\mathcal{R}}(x,\tau) = \sum_{r\in R} p_r - \sum_{r\in R} \Pr\{r \in A^h\} = \sum_{r\in R} \big(p_r - \Pr\{r \in A^h\}\big) \tag{5}$$

where the right-hand side of the equation comes from the independence hypothesis.

If we consider a request $r \in \pi_k$, the probability $\Pr\{r \in A^h\}$ only depends on the time at which vehicle k is available for r, which itself depends on previous operations. Recall the $y : R \to H$ function described in section 4: $y(r)$ is that time. Whereas $y(r)$ is deterministic for a specific scenario, it is not anymore in the context of the computation of $\Pr\{r \in A^h\}$ and we are thus interested in its probability distribution. More specifically, we compute the probability that, at a time $t \in H$, a request r already appeared and the vehicle leaves $\mathrm{w}(r)$ to satisfy it. Let's call this probability $g_1(r,t)$:

$$g_1(r,t) \equiv \Pr\{\text{request } r \text{ appeared at time } t' \leq t \text{ and } \mathrm{departureTime}(r) = t\}$$

where $\mathrm{departureTime}(r)$ is the time at which the vehicle leaves vertex $\mathrm{w}(r)$ in order to serve r, if r has been accepted. According to the recourse strategy, for a specific scenario we see that $\mathrm{departureTime}(r) = \max(y(r), t^{\min}_{r,w(r)})$.

The probability $\Pr\{r \in A^h\}$ that a request r gets satisfied is the probability that both r appears and that $\mathrm{departureTime}(r) \leq t^{\max}_{r,w}$, that is:

$$\Pr\{r \in A^h\} = \sum_{t=1}^{t^{\max}_{r,w}} g_1(r,t) = \sum_{t=t^{\min}_{r,w}}^{t^{\max}_{r,w}} g_1(r,t). \tag{6}$$

The calculus of $g_1(r,t)$ is less obvious. Since $\mathrm{departureTime}(r)$ depends on previous operations on the same waiting location $w = \mathrm{w}(r)$, we calculate $g_1(r,t)$ recursively starting from the first request $r_1 = \mathrm{fst}(\pi^w_k)$ assigned to the waiting location, up to the current request r. The second stage solution strictly respects the first stage schedule when visiting the waiting vertices, that is, these are guaranteed to be visited according to their arrival ($\underline{on}$) and departure ($\overline{on}$) times. The base case is then:

$$g_1(r_1,t) = \begin{cases} p_{r_1}, & \text{if } \ t = \max(\underline{on}(w), t^{\min}_{r_1,w}) \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Indeed, if r_1 appeared then the vehicle leaves w at time $t^{\min}_{r_1,w}$, unless it has not yet reached w at that time. The general case of a request $r >_R r_1$, $r \in \pi^w_k$, depends on the time at which the vehicle gets rid of the preceding request $\mathrm{prv}(r)$. Let $f(r,t)$ be the probability that, at time t, the vehicle either reaches back w after having served r, or discard r because it is not satisfiable or because it has revealed not to appear. It represents the time at which the vehicle becomes available for the next request after r in π^w_k, if any (computation of f is detailed below). We define $g_1(r,t)$ based on f-

probabilities of the previous request $\mathrm{prv}(r)$:

$$g_1(r,t) = \begin{cases} p_r \cdot f(\mathrm{prv}(r), t) & \text{if } \ t > t^{\min}_{r,w} \\ p_r \cdot \sum_{t'=\underline{on}(w)}^{t^{\min}_{r,w}} f(\mathrm{prv}(r), t') & \text{if } \ t = t^{\min}_{r,w} \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

Indeed, if $t > t^{\min}_{r,w}$ the vehicle leaves w to serve r as soon as it gets rid of the previous one $\mathrm{prv}(r)$. In such case, $g_1(r,t)$ is the probability that both r has already appeared and the vehicle is available for it at time t, that is, finished with request $\mathrm{prv}(r)$ at time t. At any time below $t^{\min}_{r,w}$, the probability that the vehicle leaves w must obviously be zero, since $t^{\min}_{r,w}$ is the minimum time for serving r from location $w = \mathrm{w}(r)$. At time $t = t^{\min}_{r,w}$, we must consider the possibility that the vehicle was waiting for being able to serve r, but from an earlier time $t' < t^{\min}_r$. The overall probability that the vehicle leaves w for request r at time $t = t^{\min}_{r,w}$ is then p_r times all the f-probabilities that the vehicle was actually available from a time $\underline{on}(w) \leq t' \leq t^{\min}_{r,w}$.

The f-probabilities of a request r depend on what exactly happened to r. Namely, from a time t there are two cases: either r consumed operational time, or it didn't at all:

$$f(r,t) = \ g_1(r, t - S_r) \cdot \delta^w(r, t - S_r) + \ g_1(r,t) \cdot \big(1 - \delta^w(r,t)\big) \ + \ g_2(r,t). \tag{9}$$

where $S_r = d_{w,r} + s_r + d_{r,w}$ and the function $\delta^w(r,t)$ returns 1 iff request r is satisfiable from time t and vertex w, *i.e.*, $\delta^w(r,t) = 1$ if $t \leq t^{\max}_{r,w}$, and $\delta^w(r,t) = 0$ otherwise.

The first term in the summation of the right hand side of equation (9) gives the probability that request r actually appeared and got satisfied. In such a case, $\mathrm{departureTime}(r)$ must be the current time t, minus delay S_r needed for serving r.

The second and third terms of equation (9) add the probability that the vehicle was available time t, but that request r did not consume any operational time. There are only two possible reasons for that: either r actually appeared but was not satisfiable (second term), or r did not appear at all (third term), where $g_2(r,t)$ is the probability that r did not appear and is discarded at time t, and is computed as follows. For the base case of first potential request $r_1 = \mathrm{fst}(\pi^w_k)$, we have:

$$g_2(r_1,t) = \begin{cases} 1 - p_{r_1} & \text{if } \ t = \max(\underline{on}(w), \Gamma_{r_1}) \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The general case for $r \geq r_1$, $r \in \pi^w_k$, is quite similar to the one of function g_1. We just consider the probability $1 - p_r$ that r doesn't reveal and replace $t^{\min}_{r,w}$ by Γ_r:

$$g_2(r,t) = \begin{cases} (1-p_r) \cdot f(\mathrm{prv}(r), t) & \text{if } \ t > \max(\underline{on}(w), \Gamma_r) \\ (1-p_r) \cdot \sum_{t'=\underline{on}(w)}^{\max(\underline{on}(w),\Gamma_r)} f(\mathrm{prv}(r), t') & \text{if } \ t = \max(\underline{on}(w), \Gamma_r) \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

A note on implementation. Since we are interested in computing $\Pr\{r \in A^h\}$ for each request r separately, by following the definition of g_1, we only require the f-probability associated to $\mathrm{prv}(r)$ to be already computed. This suggests a dynamic programming

Algorithm 1: Local search to compute a first stage solution of SS-VRPTW-CR

1 Let (x, τ) be an initial feasible first stage solution.
2 Initialize the neighborhood operator *op* to 1
3 **while** *some stopping criterion is not met* **do**
4 Select a solution (x', τ') at random in $\mathcal{N}_{op}(x, \tau)$
5 **if** *some acceptance criterion is met on* (x', τ') **then** set (x, τ) to (x', τ') and *op* to 1 ;
6 **else** change the neighborhood operator *op* to $op \,\%\, n_{op} + 1$;
7 **return** the best first stage solution computed during the search

approach. Computing all the f-probabilities can then be incrementally achieved while filling up a 2-dimensional matrix containing all the f-probabilities.

Computational complexity. Complexity of computing the expected cost is equivalent to the one of filling up a $|\pi_k^w| \times h$ matrix for each visited waiting location $w \in W^x$, in order to store all the $f(r, t)$ probabilities. By processing incrementally on each waiting location separately, each matrix cell can be computed in constant time using equation (9). In particular, once the probabilities in cells $(\mathrm{prv}(r), 1 \cdots t)$ are known, the cell (r, t) such that $r \neq \mathrm{fst}(\pi_k^w)$ can be computed in $\mathcal{O}(1)$ according to equations (8) - (11). Given n customer regions and a time horizon of length h, we have at most $|R| = nh \geq \sum_{w \in W^x} |\pi_k^w|$ potential requests. It then requires at most $\mathcal{O}(|R|h) = \mathcal{O}(nh^2)$ constant time operations to compute $\mathcal{Q}^{\mathcal{R}}(x, \tau)$.

6 Local Search for the SS-VRPTW-CR

Algorithm 1 describes a Simulated Annealing [23] local search approach for approximating the optimal first stage solution (x, τ), minimizing $\mathcal{Q}^{\mathcal{R}}(x, \tau)$. The computation of $\mathcal{Q}^{\mathcal{R}}(x, \tau)$ is performed according to equations of section 5 and is considered from now as a black box. Starting from an initial feasible first stage solution (x, τ), Algorithm 1 iteratively modifies it by using a set of $n_{op} = 9$ neighborhood operators. At each iteration, it randomly chooses a solution (x', τ') in the current neighborhood (line 4), and either accepts it and resets the neighborhood operator *op* to the first one (line 5), or rejects it and changes the neighborhood operator *op* to the next one (line 6). At the end, the algorithm simply returns the best solution (x^*, τ^*) encountered so far.

Initial solution and stopping criterion. The initial first stage solution is constructed by randomly adding each waiting vertex in a route $k \in [1, K]$. All waiting vertices are thus initially part of the solution. The stopping criterion depends on the computational time dedicated to the algorithm.

Neighborhood operators. We consider 4 wellknown operators for the VRP: relocate, swap, inverted 2-opt, and cross-exchange (see [24,25] for detailed description). In addition, 5 new operators are dedicated to waiting vertices: 2 for either inserting or removing from W^x a waiting vertex w picked at random, 2 for increasing or decreasing the waiting time τ_w at random vertex w $\in W^x$, and 1 that transfers a random amount of waiting time units from one waiting vertex to another.

Acceptance criterion. We use a Simulated Annealing acceptance criterion. Improving solutions are always accepted, while degrading solutions are accepted with a probability that depends on the degradation and on a temperature parameter, *i.e.*, the probability of accepting (x', τ') is $e^{-\frac{1-\mathcal{Q}^{\mathcal{R}}(x,\tau)/\mathcal{Q}^{\mathcal{R}}(x',\tau')}{T}}$. The temperature T is updated by a *cooling factor* $0 < \alpha < 1$ at each iteration of Algorithm 1: $T \leftarrow \alpha \cdot T$. During the search process, T gradually evolves from an initial temperature T_{init} to nearly zero. A restart strategy is implemented by resetting the temperature to $T \leftarrow T_{\text{init}}$ each time T decreases below a fixed limit T_{min}.

7 Experimentations

Test instances. We have randomly generated instances for the SS-VRPTW-CR. Each test instance is drawn in a square of $[100, 100]$ distance units, and is characterized by:

- The number $|C| \in \{30, 50, 80\}$ of *customer regions*, randomly distributed in the square. Each customer $c \in C$ region hosts n_{TS} potential requests.
- The number $|W| \in \{20, 30, 50\}$ of *waiting vertices*, randomly distributed in the square.
- The size $h = 480$ of the *horizon* (corresponding to the number of minutes in an 8 hour day).
- The number $n_{\text{TS}} = 24$ of *time slots*. Time slots are introduced because it is not realistic to detail request probabilities for each time unit of the horizon (*i.e.*, every minute). We set n_{TS} to 24 so that the probability that a request appears at a customer region is specified for 20 minute time slots.
- The number $K \in \{1, 3, 5, 10\}$ of available vehicles.

Travel times between vertices correspond to Euclidean distances, divided by a velocity parameter specified in the instance. Figure 3 shows an example of test instance. As a convention, the first time slot is associated to time unit 1 whereas time slot i is associated to time unit $1 + (i-1) \cdot \lfloor h/n_{\text{TS}} \rfloor$. In our instance *a.1*, a potential request r associated to time slot 2 has a reveal time $\Gamma_r = 21$ and no potential request is associated to time units $[2, 20]$, $[22, 40]$, etc. All the test instances are available at `http://becool.info.ucl.ac.be/resources/benchmarks-ss-vrptw-cr`.

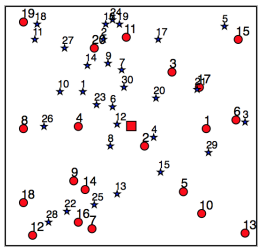


Fig. 3. Map representation of instance $a.1$. The depot (square) is located at the center. The instance counts 30 customer regions (stars) and 20 waiting vertices (circles).Although it is not visible here, the instance has a time horizon of 480 units and counts 24 time slots. If the operational day lasts 8 hours, a time unit represents a 1 minute in real time and each time slot lasts 20 minutes.

Potential requests. Each potential request r, associated with a customer region and a time slot, comes with a deterministic service time $s_r = 10$. The time window $[e_r, l_r]$ is such that e_r is chosen uniformly in $[\Gamma_r, \min(\Gamma_r + \frac{h}{n_{\text{TS}}}], h)$, and l_r is chosen uniformly in $[\max(e_l, d_{0,r}), \max(e_l + 10, d_{0,r})]$.

Scale parameter. A *scale* parameter is introduced in order to optimize expectations on coarser data, and therefore to speed-up computations. When equal to 1, expectations are computed while considering the original horizon. When *scale* > 1, expectations are computed from a coarse version of the initial horizon, scaled down by the factor $scale$. If *scale* =10 for instance, then the horizon is scaled to $h' = 48$. All the time data, such as travel and service times, but also time windows and reveal times, are then scaled as well (rounding up to nearest integer). When working on a scaled horizon (i.e. scale > 1), Algorithm 1 deals with an approximate but easier objective function $\mathcal{Q}^{\mathcal{R}}(x, \tau)$, in $\mathcal{O}(n(\frac{h}{\text{scale}})^2)$, and a reduced search space due to a coarse time horizon.

Algorithm 1 is then modified by simply adapting line 7 to return the best solution encountered so far, *according to the initial horizon*. Each time a new best solution is found during the search, its true expected cost is computed after scaling it up back to the original horizon, multiplying arrival, departure and waiting times by a factor *scale*.

Experimental plan. All experiments are done under a cluster composed of 32 64-bits AMD Opteron 1.4GHz cores. The code is developed in *C++11* and compiled with LLDB using *-O3* optimization flag. The Simulated Annealing parameters are set to $T_{\text{init}} = 5$, $T_{\text{min}} = 10^{-6}$ and $\alpha = 0.995$.

	30 seconds				3 minutes				30 minutes			
scale:	**1**	**2**	**5**	**10**	**1**	**2**	**5**	**10**	**1**	**2**	**5**	**10**
a.1	19.7	18.2	**15.0**	16.2	16.7	16.4	**14.9**	16.2	16.1	15.3	**14.9**	16.5
a.2	22.2	20.1	**16.4**	17.5	17.7	16.8	**16.2**	17.4	16.7	16.3	**16.2**	17.5
a.3	20.3	20.0	**14.4**	16.1	16.1	15.9	**14.0**	16.0	15.6	15.2	**14.0**	16.2
b.1	21.1	16.9	11.1	**11.0**	8.2	**6.3**	7.1	9.9	**5.6**	5.7	6.9	9.6
b.2	22.1	17.5	**9.6**	11.4	5.6	7.7	**6.8**	9.9	**5.1**	7.4	6.4	9.3
b.3	22.2	17.0	**10.8**	11.9	8.7	**7.2**	7.8	11.1	**6.2**	8.3	7.2	10.8
c.1	42.1	38.7	**23.6**	25.9	15.3	13.9	**12.2**	19.3	**8.3**	9.3	11.0	16.7
c.2	43.9	37.2	**25.8**	27.8	14.0	14.9	**13.5**	19.8	**9.9**	10.4	11.6	17.9
c.3	42.4	39.2	**24.3**	24.8	17.5	14.9	**11.7**	17.5	15.2	**8.8**	10.3	15.8
d.1	71.6	67.9	54.8	**54.3**	46.5	30.4	**26.1**	39.6	**11.8**	13.0	19.2	32.7
d.2	72.2	67.9	**52.8**	56.2	40.9	34.7	**25.7**	40.1	**12.6**	19.0	20.3	31.4
d.3	73.0	67.4	53.8	**51.5**	40.8	37.2	**23.0**	35.9	17.4	**11.9**	18.4	28.4

Table 1. Experimental results while varying horizon scale and computational time.

Results. Table 1 shows average experimental results over 10 runs on 12 instances: Instances $a.x$ (resp. $b.x$, $c.x$ and $d.x$) are such that $|C| = 30$ (resp. 30, 50 and 80), $|W| = 5$ (resp. 20, 30 and 50), and $K = 1$ (resp. 3, 5 and 10). Results are reported with *scale* $\in \{1, 2, 5, 10\}$ and with a CPU time limit $\in \{30, 180, 1800\}$ seconds.

Provided a limited computational time of 30 seconds, using a scaled horizon leads to better results. This is easily explained by the limited number of local search iterations performed when *scale*=1. As the computational time increases to 3 minutes, working on the original horizon size tends to provide better results. This trend is confirmed by moving to 30 minutes. As the available computational time increases, the accuracy in the objective function eventually overtakes the computational efficiency provided by scaled horizons, especially for large instances such as $c.x$ and $d.x$.

With their unique vehicle and because requests are uniformly distributed, instances $a.x$ may suffer from an evaluation function being roughly uniform. Sending the vehicle at some location to wait there is, most of the time, more or less equivalent to another location. Consequently, local optima are numerous and little diversified, the more promising ones being hard to detect when using scale 1 for only 30 minutes. In the contrary, using more vehicles (e.g. instances $b.x$) leads to a less uniform evaluation function. For example, concentrating all the vehicles in the same region would surely leads to poor results. On instances $a.x$, the diversification brought by scaled horizons then still prevails after 30 minutes. Given larger computation times (5 hours), results on scales 5 and 10 do not show a significant improvement:

	a.1				a.2				a.3			
scale $\in \{1, 2, 5, 10\}$:	15.3	15.1	**14.9**	16.5	16.2	**15.8**	16.1	17.3	14.8	14.6	**14.0**	16.1

Scales 1 and 2 in contrary tend to take promising benefits of a larger computation time.

Figure 4 shows how, for instance *c.1*, the real objective function (i.e. according to the original horizon) evolves in average (over 10 runs) during an execution of Algorithm 1. By reducing both the granularity of the search space and the complexity of the objective function, the parameter *scale* can therefore be used as a tradeoff between responsiveness and good quality solutions on the long term. Figure 4 also shows that the parameter *scale* can dynamically be reduced during the search.

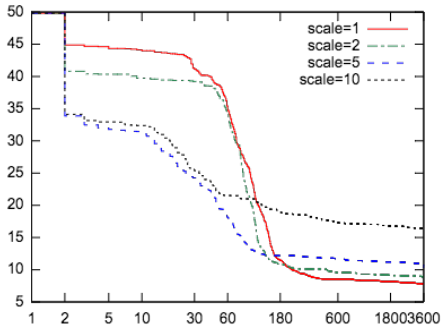


Fig. 4. Average evolution of the best real objective value in Algorithm 1, during 3600 seconds on instance $c.1$. During the first second, objective values rapidly decrease when optimizing on scaled horizon. Thereafter, depending on the available computation time, some scale factors reveal to be more efficient than others. For less than 1 minute, *scale*=5 leads to better results. With at least 10 minutes, using the original horizon is definitely better.

8 Conclusions and research directions

We introduced a new stochastic VRP, the SS-VRPTW-CR. Unlike existing SS-VRPs with random customers, we don't make any assumption on the moment at which a customer reveals its presence or absence. Instead, this is treated as a random variable as well. We proposed a recourse strategy for a special case of the SS-VRPTW-CR, when there is no maximal vehicle capacities. We showed how the exact expected cost can be computed in pseudo-polynomial time under this recourse strategy, and how to integrate in an efficient meta-heuristic method. Experiments are driven on generated test instances of various sizes. The average results show how a scale parameter, controlling the granularity of the time horizon, can be used to tune the optimization process in the case of limited computational times.

Maximal vehicle capacity constraints. The recourse strategy and equations we give can be extended to take care of vehicle capacities. We are currently working on a generalized version of these equations.

Contribution to online optimization. Another potential application of the SS-VRPTW-CR goes to online optimization problems such as the DS-VRPTW. Because of the huge complexity of reoptimization, heuristic methods are often preferred, including the so called Sample Average Approximation (SAA, see [26]). SAA relies on Monte Carlo sampling, making decisions based on a subset of the scenarios. Thanks to recourse strategies, the SS-VRPTW-CR provides an upper bound on the expected cost of a first stage solution under optimal reoptimization. The SS-VRPTW-CR could therefore be used as a subroutine in order to heuristically solve the DS-VRTPW, whilst considering the whole set of scenarios instead of only a subset of sampled ones. In such a context, the scale parameter we introduce in the experiments can be of great contribution.

Acknowledgments Christine Solnon is supported by the LABEX IMU (ANR-10-LABX-0088) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

References

1. Bent, R.W., Van Hentenryck, P.: Waiting and Relocation Strategies in Online Stochastic Vehicle Routing. IJCAI (2007) 1816–1821
2. Ichoua, S., Gendreau, M., Potvin, J.Y.: Exploiting knowledge about future demands for real-time vehicle dispatching. Transportation Science **40**(2) (may 2006) 211–225
3. Saint-Guillain, M., Deville, Y., Solnon, C.: A Multistage Stochastic Programming Approach to the Dynamic and Stochastic VRPTW. In: International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems. (2015) 357–374
4. Branke, J., Middendorf, M., Noeth, G., Dessouky, M.: Waiting Strategies for Dynamic Vehicle Routing. Transportation Science **39**(3) (aug 2005) 298–312
5. Bertsimas, D.J.: A vehicle routing problem with stochastic demand. Operations Research (1992)

6. Jaillet, P.: Probabilistic traveling salesman problems. PhD thesis, Massachusetts Institute of Technology (1985)
7. Laporte, G., Louveaux, F.V., Mercure, H.: A priori optimization of the probabilistic traveling salesman problem. Operations Research **42**(3) (1994) 543–549
8. Laporte, G., Louveaux, F.V.: The integer L-shaped method for stochastic integer programs with complete recourse. Oper. Res. Lett. **13**(3) (1993) 133–142
9. Jezequel, A.: Probabilistic vehicle routing problems. PhD thesis, Massachusetts Institute of Technology (1985)
10. Bertsimas, D.J., Chervi, P., Peterson, M.: Computational approaches to stochastic vehicle routing problems. Transportation science (1995) 1–34
11. Bianchi, L., Campbell, A.M.: Extension of the 2-p-opt and 1-shift algorithms to the heterogeneous probabilistic traveling salesman problem. European Journal of Operational Research **176**(1) (jan 2007) 131–144
12. Bowler, N.E., Fink, T.M., Ball, R.C.: Characterisation of the probabilistic travelling salesman problem. Physical Review E **68**(3) (nov 2003)
13. Bianchi, L., Gambardella, L.M., Dorigo, M.: An ant colony optimization approach to the probabilistic traveling salesman problem. Parallel Problem Solving from Nature (2002) 883–892
14. Waters, C.D.J.: Vehicle-scheduling problems with uncertainty and omitted customers. Journal of the Operational Research Society (1989) 1099–1108
15. Gendreau, M., Laporte, G., Séguin, R.: An exact algorithm for the vehicle routing problem with stochastic demands and customers. Transportation Science **29**(2) (1995) 143–155
16. Séguin, R.: Problemes stochastiques de tournees de vehicules. Centre de Recherche sur les Transports Publication **979** (1994)
17. Gendreau, M., Laporte, G., Séguin, R.: A Tabu Search Heuristic for the Vehicle Routing Problem with Stochastic Demands and Customers. Operations Research **44**(3) (1996) 469–477
18. Gounaris, C.E., Repoussis, P.P., Tarantilis, C.D., Wiesemann, W., Floudas, C.A.: An adaptive memory programming framework for the robust capacitated vehicle routing problem. Transportation Science (2014)
19. Campbell, A.M., Thomas, B.W.: Probabilistic traveling salesman problem with deadlines. Transportation Science **42**(1) (2008) 1–27
20. Henchiri, A., Bellalouna, M., Khaznaji, W.: A probabilistic traveling salesman problem: a survey. In: FedCSIS Position Papers. Volume 3. (sep 2014) 55–60
21. Sungur, I., Ren, Y.: A model and algorithm for the courier delivery problem with uncertainty. Transportation science **44**(2) (2010) 193–205
22. Chao, I.M., Golden, B.L., Wasil, E.A.: The team orienteering problem. European Journal of Operational Research **88**(3) (1996) 464–474
23. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598) (1983) 671–680
24. Kindervater, G.A.P., Savelsbergh, M.W.P.: Vehicle routing: handling edge exchanges. Local search in combinatorial optimization (1997) 337–360
25. Taillard, É., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.Y.: A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation science **31**(2) (1997) 170–186
26. Ahmed, S., Shapiro, A.: The sample average approximation method for stochastic programs with integer recourse. Submitted for publication (2002)

Apprentissage de clauses nobetters dans les solveurs séparation et évaluation pour Max-SAT

André Abramé **Djamal Habet**

Aix-Marseille Université, Université de Touloun, CNRS, ENSAM, LSIS, Marseille, France
{andre.abrame,djamal.habet}@lsis.org

Résumé

Nous introduisons une nouvelle méthode d'apprentissage de clauses dites *nobetters* pour les solveurs séparation et évaluation pour Max-SAT. Elle s'inspire de l'apprentissage de clauses *nogoods* utilisé par les solveurs SAT basés sur l'analyse de conflits (CDCL). Elle a pour objectif de permettre une meilleure résolution des instances industrielles par une meilleure prise en compte de leurs structures. Ce travail est un résumé d'un article que nous avons publié à la conférence ICTAI 2016 [1].

1 Introduction

Le problème Max-SAT consiste à déterminer, pour une formule propositionnelle en forme normale conjonctive, une interprétation des variables de cette formule maximisant le nombre de clauses satisfaites.

Les solveurs de type séparation et évaluation (S&E) pour Max-SAT ont montré leur efficacité sur les instances aléatoires et *crafted*. Cependant, sur les instances fortement structurées (comme celles issues de problèmes industriels), ils sont sensiblement moins performants que d'autres types de solveurs (notamment basés sur des appels itératifs à un solveur SAT moderne). Cela peut s'expliquer par leur incapacité à exploiter les informations structurelles de ces instances. À l'inverse, les solveurs modernes CDCL (Conflict-Driven Clause Learning) pour SAT sont très efficaces sur les instances industrielles. Une des raisons expliquant cette efficacité est le mécanisme d'apprentissage des clauses *nogoods*.

Dans ce papier, nous proposons un nouveau mécanisme d'apprentissage pour les solveurs S&E inspiré de celui des solveurs modernes. Il est basé sur la notion de clauses *nobetters*. Leur but est d'interdire les interprétations partielles dont les extensions ne permettront pas d'améliorer la meilleure solution trouvée jusqu'à présent. L'apprentissage des clauses *nobetters* permet de transposer aux solveurs S&E pour Max-SAT des techniques associées aux *nogoods* dans les solveurs modernes pour SAT.

2 Apprentissage de clauses nogoods dans les solveurs SAT

L'idée sous-jacente à l'apprentissage de clauses *nogoods* consiste à identifier les causes des conflits (i.e. un sous-ensemble de l'interprétation courante ayant mené au conflit) en analysant les étapes d'applications des règles d'inférences sémantiques utilisées pour étendre l'interprétation courante. Pour éviter la répétition de cette même interprétation, qui mènerait au même conflit, une clause (la clause *nogood*) est ajoutée à la formule. Ainsi, cela permet de limiter la redondance dans l'exploration de l'arbre de recherche tout en détectant plus tôt les conflits. De plus, ces clauses et les informations déduites durant leur construction sont utilisées par les solveurs modernes pour guider l'exploration de l'espace de recherche.

3 Apprentissage de clauses nobetters dans les solveurs Max-SAT

Comme les solveurs SAT modernes, les solveurs S&E pour Max-SAT explorent l'espace de recherche en construisant un arbre de recherche. Ils maintiennent deux valeurs : la borne supérieure (UB), qui est la meilleure solution trouvée jusqu'à présent et la borne inférieure (LB) qui correspond à une (sous-)estimation de la meilleure solution accessible dans la branche courante de l'arbre de recherche. À un nœud donné de l'arbre, si LB $\geq$ UB alors ils effectuent un retour-arrière (*backtrack*) jusqu'à atteindre une branche in-

explorée de l'arbre de recherche.

Lorsqu'un conflit est détecté (i.e. que LB $\geq$ UB), on sait que le sous-ensemble de l'interprétation courante qui apparaît dans les clauses falsifiées ne peut être étendu à une interprétation complète améliorant la meilleure solution connue. Comme dans le cadre des *nogoods* pour SAT, on peut alors analyser la séquence d'application des règles d'inférence sémantiques pour identifier un sous-ensemble de l'interprétation courante ayant mené au conflit, puis ajouter une nouvelle clause dure interdisant cette interprétation : la clause *nobetter*.

Pour mettre en place un tel système, il est nécessaire de redéfinir la notion de conflit. En effet, contrairement à SAT où il suffit d'une clause falsifiée pour rendre une interprétation conflictuelle, dans le cadre de Max-SAT il faut considérer toutes les clauses falsifiées ayant participé au calcul de la borne inférieure. De plus, les règles d'inférence sémantiques utilisées par les solveurs S&E pour Max-SAT ne sont pas les mêmes que dans le cadre de SAT. Nous avons donc défini, pour chacune d'entre elles (propagation unitaire réelle, règle des littéraux purs et règle des clauses unitaires dominantes), les causes provoquant leur application. Ces deux éléments sont suffisants pour construire un graphe d'implications décrivant la séquence d'application des règles d'inférence sémantiques ayant mené à l'interprétation courante. L'analyse de ce graphe, en partant des clauses falsifiées, permet de construire la clause *nobetter*.

Comme indiqué ci-dessus, plusieurs clauses falsifiées doivent être analysées pour produire une clause *nobetter*. Par conséquent, les clauses *nobetters* apprises seront potentiellement plus grandes que dans le cadre des solveurs SAT. Pour y remédier, nous avons modifié la procédure de construction des clauses *nobetters* en poursuivant l'analyse des causes des affectations jusqu'à atteindre le point d'implication unique de chaque niveau de décision apparaissant dans la clause apprise.

Enfin, il est intéressant de noter que la plupart des techniques utilisées dans les solveurs SAT modernes en combinaison avec l'apprentissage des clauses *nogoods* peuvent être adaptées aux solveurs S&E pour Max-SAT utilisant l'apprentissage des *nobetters*. En particulier, on peut effectuer des retours-arrières non-chronologique (*backjump*), des redémarrages ou encore mettre en place des heuristiques de branchement basées sur l'apparition des variables dans les conflits. L'intégration de ces techniques n'est cependant pas triviale et nécessitera une étude approfondie des interactions avec les autres composants des solveurs S&E.

4 Résultats expérimentaux

Nous avons évalué expérimentalement l'impact de l'apprentissage des clauses *nobetters* dans notre solveur AHMAXSAT. Les tests sont réalisés sur les instances et en suivant le protocole de la Max-SAT Evaluation 2015.

D'une part, les résultats obtenus montrent que le calcul des clauses *nobetters* entraîne une augmentation de 20% du temps moyen de résolution à cause du maintien des informations nécessaires à leur calcul. Par ailleurs, sur les instances industrielles, l'apprentissage des clauses *nobetters* permet de réduire sensiblement le nombre de décisions nécessaires pour résoudre les instances (-15% sur les instances industrielles Max-SAT partielles et -54% sur le même type d'instances mais pondérées). Par conséquent, plus d'instances sont résolues (231 contre 217 pour la version originale d'AHMAXSAT). Cependant, le temps moyen de résolution sur ces instances est dans l'ensemble inchangé, le surcoût induit par le calcul des clauses *nobetters* réduisant le gain obtenu par la diminution du nombre de décisions.

Enfin, malgré cette amélioration des performances sur les instances industrielles, les résultats obtenus restent encore en deçà de ceux des solveurs Max-SAT les plus performants sur ces catégories d'instances.

5 Conclusion

Nous avons introduit un nouveau mécanisme d'apprentissage de clauses inspiré de celui utilisé par les solveurs SAT modernes. Nous avons posé les bases théoriques de ce mécanisme et présenté les algorithmes permettant sa mise en œuvre. Les résultats expérimentaux que nous avons obtenus montrent que l'apprentissage des clauses *nobetters* améliore sensiblement les performances de notre solveur sur les instances industrielles. Ce travail est une étape et mérite d'être poursuivi. En effet, notre implémentation de l'apprentissage des clauses *nobetters* remplit seulement un de ses objectifs : limiter la redondance lors de la recherche. Il serait intéressant d'implémenter d'autres techniques utilisées dans les solveurs SAT modernes, telles que les redémarrages ou des heuristiques de branchement basées sur l'activité des variables.

Références

[1] A. Abramé and D. Habet. Learning Nobetter Clauses in Max-SAT Branch and Bound Solvers. *Proceedings of the 28th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2016)*, pages 452–459, 2016.

Filtrage tardif du BIBD : lorsque la procrastination paie

Yassine Attik **Jonathan Gaudreault** **Claude-Guy Quimper**

Consortium de recherche FORAC, Université Laval, Québec, Canada

Yassine.Attik@cirrelt.ca {Jonathan.Gaudreault,Claude-Guy.Quimper}@ift.ulaval.ca

Résumé

Lorsque nous développons une nouvelle contrainte pour un solveur de programmation par contraintes, il faut implémenter les algorithmes de filtrage qui se déclenchent durant la recherche lorsque surviennent différents événements (e.g. *une variable a été instanciée*, *une valeur a été retirée du domaine*, etc.). Dans le cadre de cette recherche, nous cherchons à savoir quel est le meilleur moment pour filtrer lorsque nous tentons de résoudre le problème du *Balanced Incomplete Block Design* (BIBD). Retarder la majorité du filtrage jusqu'au dernier moment, ce qui revient à dire qu'on filtre uniquement lorsque la variable vient d'être sélectionnée et est sur le point d'être instanciée, réduit le temps de calcul entre 20 % et 30 %. L'approche préserve le même espace de solution et le même niveau de filtrage. Les solveurs génériques n'offrent pas de manière explicite la possibilité de rattacher le filtrage à l'événement *la variable vient d'être sélectionnée*. Nos travaux indiquent cependant que cela pourrait être intéressant de l'inclure dans les solveurs génériques. Bien sûr, l'approche ne garantit pas le même niveau de filtrage pour tous les problèmes. À cet effet, nous présentons un contre-exemple.

Abstract

When developing a new constraint for a CP solver, one must implement filtering algorithms that trigger during the search according to different events (e.g. *a variable is instantiated*, *a value is removed from the domain*, etc.). In the context of this research, we studied what is the best moment to carry on filtering while solving the Balanced Incomplete Block Design (BIBD) problem. It showed that delaying most of the filtering until the last moment that is, filtering the domain of a variable only when the variable is selected and is about to be instantiated, reduces computation time by 20 % to 30 %. The approach preserved the same solution space and the same level of filtering. Current generic solvers do not provide a *variable is selected* event to attach filtering algorithms to. It could be interesting to add this option into generic solvers. Of course, our approach does not guarantee the same filtering level for all the problems, which we illustrate with a counter-example.

1 Introduction

Le problème du Balanced Incomplete Block Design (BIBD) possède plusieurs applications importantes telles que le design de plans d'expérience, la planification d'horaire de tournoi, le *coding theory* [6, 17, 18] et l'optimisation de portfolios financiers [2]. Il peut être résolu en utilisant différentes approches telles que la recherche locale [9], la programmation en nombres entiers [18] ou la programmation par contraintes (PPC) [8, 10].

Dans le contexte de cette recherche, nous étudions quel est le meilleur moment pour effectuer le filtrage lorsque nous résolvons un BIBD en utilisant les techniques de PPC. Lorsqu'on développe une nouvelle contrainte pour un solveur de PPC, il faut aussi implémenter les algorithmes de filtrage qui seront déclenchés durant la recherche lorsque différents événements vont survenir. Les événements les plus communs sont l'instanciation d'une variable, la modification d'une borne (borne supérieure ou inférieure) ou, plus généralement, le retrait d'une valeur [12, 15, 16].

Nous essayons une autre approche que nous appelons le filtrage tardif (*lazy filtering*). Nous retardons le filtrage du domaine d'une variable jusqu'au dernier moment, c'est-à-dire, lorsque la variable a été sélectionnée et qu'elle est sur le point d'être instanciée. Il apparaît que, pour les BIBDs, cette approche offre exactement le même niveau de filtrage tout en réduisant le temps de résolution de 20 % à 30 %. Bien sûr, le résultat ne peut pas être généralisé à tous les problèmes en PPC, ce que nous illustrons avec un contre-exemple qui n'est pas un BIBD.

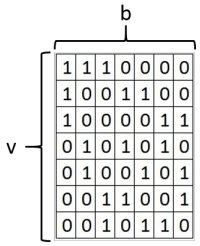


Image 1 – Représentation binaire

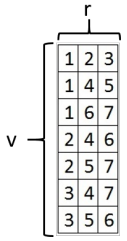


Image 2 – Représentation n-aire

Le reste de cet article est organisé ainsi. La section 2 présente les concepts préliminaires à propos des BIBDs, la PPC et le filtrage. La section 3 décrit comment les contraintes du BIBD peuvent être adaptées pour le filtrage tardif. La section 4 montre les résultats des expériences. La section 5 conclut l'article.

2 Concepts préliminaires

2.1 Balanced Incomplete Block Design

Un problème BIBD est défini par un tuple (v, b, r, k, λ). Nous avons v types d'objets qui doivent être assignés à b blocs de manière à ce que chaque bloc contienne k objets de différents types. Chaque type d'objets est présent dans r blocs différents. Pour n'importe quel pair de type d'objets, ils doivent apparaître simultanément dans λ différents blocs [1].

Par définition, les instances BIBDs doivent satisfaire les propriétés suivantes [17] :

1. $b = \frac{\lambda(v^2-v)}{k^2-k}$
2. $r = \frac{\lambda(v-1)}{k-1}$

De ce fait, une instance BIBD peut être définie en utilisant seulement les paramètres (v, k, λ).

Le problème peut être modélisé en utilisant une matrice de variables binaires ayant v lignes et b colonnes $X = (x_{ij}) \in \{0,1\}^{v \times b}$, où x_{ij} est égal à 1 *ssi* un objet du type i est assigné au bloc j. Il peut aussi être modélisé en utilisant une matrice $Y = (y_{ij}) \in [1..b]^{v \times r}$ où pour chaque ligne, correspondant à un type d'objet i, nous indiquons dans quels r blocs de 1 à b ils sont placés. Ces deux représentations, mathématiquement équivalentes, sont illustrées par les images 1 et 2 qui montrent une même solution pour l'instance $(7, 3, 1)$.

Les trois premières lignes du Tableau 1 sont les contraintes requises pour représenter un BIBD sous la forme d'une matrice (versions binaire et n-aire). La représentation d'un BIBD sous forme matricielle amène de la symétrie qui est importante à considérer. En effet, pour n'importe quelle solution que nous trouvons, il existe $v! \times b!$ autres solutions symétriques [1, 4]. C'est pourquoi il est important d'utiliser des stratégies de bris de symétrie comme celles indiquées par les contraintes 4 et 5 du Tableau 1.

1. http ://www.csplib.org/Problems/prob028

2.2 Programmation par contraintes, filtrage et propagation

En PPC, chaque contrainte possède des algorithmes de filtrage dont le travail est de retirer les valeurs des domaines qui ne satisfont pas la contrainte.

Les algorithmes de filtrage sont déclenchés lorsqu'un domaine est modifié. Pour une contrainte donnée, un ou plusieurs algorithmes peuvent être associés à différents événements qui correspondent à différentes modifications qui sont survenues sur le domaine [16]. Les événements généralement reconnus par les solveurs sont : l'instanciation d'une variable, la modification d'une borne inférieure, la modification d'une borne supérieure et le retrait d'une valeur [12, 15, 16]. Schulte et Stuckey [15] proposent d'autres événements tels que lorsque toutes les valeurs d'un domaine deviennent positives ou lorsqu'une valeur spécifique est retirée du domaine. Les différents algorithmes de filtrage sont exécutés de manière répétitive (propagation) jusqu'au moment où aucun d'entre eux ne peut plus modifier de domaines davantage. Nous atteignons donc un *fixpoint* [12, 15, 16].

Il existe des situations particulières où 2 niveaux de filtrage deviennent équivalents [14]. Dans ces cas-là, il est préférable d'utiliser l'algorithme le plus rapide.

Avoir le même niveau de filtrage signifie que nous visitons les mêmes solutions ordonnées de la même manière tout en ayant un nombre d'échecs identique lors de l'exploration des arbres. Cela veut donc dire que nous avons exploré le même arbre de recherche pour une version ou une autre.

3 Filtrage tardif des BIBDs

Il existe généralement un compromis à faire entre le temps passé à explorer un arbre de recherche et le

Tableau 1 – Contraintes pour BIBD binaire vs n-aire. Les contraintes GCC et NValue sont définies dans [13] et [7] respectivement. La contrainte GCC est définie comme étant $GCC([X_1...X_n],[l_1...l_m],[u_1...u_m])$ où $\forall x \quad l_x \leq |\{i \mid X_i = x\}| \leq u_x$. La contrainte NValue est définie comme étant $NValue([X_1...X_n],x) \iff |\{X_1...X_n\}| = x$ GCC et NValue sont implémentés de manière simplifiée dans 3.2 ainsi que 3.3 considérant l'ordonnancement statique des variables que nous utilisons.

	Binaire	**n-aire**
1. Chaque type d'objet doit apparaître dans r blocs	$\sum_{j=1}^{b} x_{ij} = r \quad \forall i \in [1..v]$	$y_{ij} < y_{i,(j+1)} \quad \forall i \in [1..v]$ $\forall j \in [1..(r-1)]$ Cette contrainte brise partiellement la symétrie de colonnes.
2. Chaque bloc doit contenir exactement k objets distincts	$\sum_{i=1}^{v} x_{ij} = k \quad \forall j \in [1..b]$	$GCC(Y,[k_1,\ldots,k_b],[k_1,\ldots,k_b])$
3. Deux objets distincts doivent apparaître ensemble dans λ blocs	$\sum_{j=1}^{b} x_{ij}x_{lj} = \lambda$ $\forall i,l \in [1..v]$ *et* $i < l$	$NValue([y_{i1},\ldots,y_{ir},y_{l1},\ldots,y_{lr}],$ $2r-\lambda) \quad \forall 1 \leq l < i \leq v$
4. Ordre lexicographique sur les lignes	$x_{i,1},\ldots,x_{i,b} \prec_{\text{LEX}}$ $x_{(i+1),1},\ldots,x_{(i+1),b}$ $\forall i \in [1..(v-1)]$	$y_{i,1},\ldots,y_{i,r} \prec_{\text{LEX}} y_{(i+1),1},\ldots,y_{(i+1),r}$ $\forall i \in [1..(v-1)]$
5. Ordre lexicographique sur les colonnes	$x_{1,j},\ldots,x_{v,j} \succeq_{\text{LEX}}$ $x_{1,(j+1)},\ldots,x_{v,(j+1)}$ $\forall j \in [1..(b-1)]$	Nous pouvons facilement implémenter une heuristique de choix de valeur qui fait cela. Si nous avons le choix entre plusieurs valeurs différentes qui ont, jusqu'à maintenant, été utilisées sur les mêmes lignes, alors nous utilisons la plus petite valeur possible.

temps passé à filtrer cet arbre en utilisant les algorithmes de filtrage. Ce n'est pas le sujet de notre recherche présentée ici. Plutôt, nous montrons que, pour un BIBD, il est possible, en remettant le filtrage au dernier moment, de conserver le même niveau de filtrage tout en diminuant le temps de résolution.

Nous filtrons le domaine d'une variable au tout dernier moment, c'est-à-dire, entre le moment où la variable est sélectionnée et juste avant que l'heuristique de choix de valeur soit appelée. De ce fait, nous ajoutons un événement de filtrage au solveur qui s'appelle *variableAÉtéSélectionnée*. C'est une pratique plutôt inhabituelle puisque les solveurs préfèrent filtrer le plus tôt possible, c'est-à-dire, au moment où le domaine d'une autre variable est modifié.

Les sections 3.1 à 3.5 présentent comment les algorithmes de filtrages pour le BIBD peuvent être implémentés. Il est important de remarquer qu'un algorithme est différent selon que l'algorithme de filtrage se déclenche tardivement (*variableAÉtéSélectionnée*) ou avec les événements classiques (bornes de la variable modifiées ou variable instanciée). Cette différence est due au fait que, dans la version traditionnelle, nous pouvons modifier des domaines de n'importe quelle variable de la matrice, alors qu'en version tardive, nous modifions que le domaine de la variable qui vient d'être sélectionnée. Dans tous les cas, nous obtenons le même niveau de filtrage, mais avec des temps de résolution différents que nous allons évaluer dans la section 4.

Nous présentons les algorithmes et les résultats d'expériences pour la représentation n-aire des BIBD puisqu'ils étaient plus faciles à adapter avec notre technique. Nous donnons aussi l'analyse de la complexité des algorithmes. Des algorithmes équivalents existent pour le modèle binaire. L'implémentation des algorithmes de filtrage décrits suppose que les variables sont instanciées selon une séquence statique (de la gauche vers la droite, du haut vers le bas). Il s'agit d'une pratique courante pour la résolution du BIBD [2]. De même, nous instancions les variables en choisissant d'abord les plus petites valeurs (il s'agit cependant d'un choix arbitraire puisque, par définition, les valeurs ne sont pas réellement « numériques », mais

2. Choco [11] et Gecode [5] utilisent des heuristiques statiques pour la résolution du BIBD dans les fichiers exemples de leurs solveurs.

correspondent en réalité à des identifiants de blocs où on souhaite placer les objets).

3.1 Chaque type d'objet doit apparaître dans r blocs

Cette contrainte force les valeurs d'une ligne à prendre une valeur différente. On y arrive en imposant un ordre croissant. Cela permet aussi, partiellement, de briser une forme de symétrie.

3.1.1 Filtrage traditionnel

Dans une ligne i où la variable y_{ij} a été instanciée, nous filtrons le domaine de toutes les variables $l > j$ en utilisant la formule (1). La complexité de cet algorithme est $O(r)$.

$$y_{ij} < y_{il} \qquad l \in [(j+1)..r] \tag{1}$$

3.1.2 Filtrage tardif

Lorsque la variable y_{ij} est sélectionnée, nous avons seulement besoin de retirer du domaine courant les valeurs inférieures à la valeur de $y_{i,(j-1)}$ en utilisant la formule (2). La complexité de cet algorithme est $O(1)$.

$$y_{i,(j-1)} < y_{ij} \tag{2}$$

Avec cette version, chaque variable de la ligne est filtrée qu'une seule fois, alors qu'avec le filtrage traditionnel, les variables de la dernière colonne sont filtrées jusqu'à $r-1$ fois. Cependant, pour les deux algorithmes, le même nombre de valeurs sont retirées.

3.2 Chaque bloc doit contenir exactement k objets distincts

Cette contrainte requiert que chaque valeur apparaisse k fois dans la matrice. Dans la version traditionnelle, nous devons vérifier l'ensemble de la matrice lorsqu'une variable est instanciée. Dans la version tardive, nous avons seulement besoin de vérifier les variables précédemment instanciées.

3.2.1 Filtrage traditionnel

Lorsqu'une variable y_{ij} est instanciée, nous devons vérifier, pour toute la matrice, le nombre de fois que la valeur de y_{ij} est utilisée. Si la valeur a été utilisée k fois, alors nous devons retirer du domaine des variables qui suivent la valeur de y_{ij}. La complexité de cet algorithme est $O(vrb)$.

3.2.2 Filtrage tardif

Lorsque la variable y_{ij} est sélectionnée, nous devons vérifier, seulement pour les variables précédemment instanciées, combien de fois chaque valeur est instanciée. Par la suite, nous retirons du domaine de la variable y_{ij} les valeurs qui sont utilisées k fois. La complexité de cet algorithme, en pire cas, est $O(vr+b)$.

3.3 Deux objets distincts doivent simultanément apparaître dans λ blocs

Dans la version traditionnelle, il est nécessaire de vérifier les $v-1$ paires de lignes que la ligne courante peut former avec les autres lignes. Dans la version tardive, il est seulement nécessaire de vérifier les paires de lignes que la ligne courante peut former avec les lignes précédentes.

3.3.1 Filtrage traditionnel

À partir de la ligne i à laquelle la variable y_{ij} qui vient d'être instanciée appartient, nous devons considérer les $v-1$ paires de lignes formées avec les autres lignes.

Pour chaque paire, nous devons calculer, pour les variables déjà instanciées, le nombre de fois que chaque valeur est utilisée. Si pour une paire nous avons exactement λ valeurs qui sont utilisées deux fois, alors nous devons retirer de chaque ligne qui forment la paire les valeurs des variables instanciées de l'autre ligne. Si nous avons plus que λ valeurs utilisées deux fois, alors il faut forcer un échec. La complexité de cet algorithme est $O(vrb)$.

3.3.2 Filtrage tardif

Lorsque la variable y_{ij} est sélectionnée, nous devons, pour chaque $i-1$ paires qu'on peut former avec les lignes précédentes, calculer le nombre de fois que chaque valeur est utilisée. Si nous avons exactement λ valeurs utilisées deux fois, alors nous devons retirer du domaine de la variable sélectionnée y_{ij} les valeurs des variables instanciées de l'autre ligne de la paire. La complexité de cet algorithme est $O(r+b)$ dans le meilleur cas lorsqu'on filtre la première ligne avec la deuxième et le pire cas est $O(vr+b)$ lorsqu'on filtre la dernière ligne avec les autres au-dessus.

3.4 Ordre lexicographique entre les lignes

3.4.1 Filtrage traditionnel

Pour le filtrage traditionnel, nous utilisons l'algorithme d'ordre lexicographique strict de Frisch et coll. [3] qui se déclenche lorsqu'une borne

(supérieure ou inférieure) d'une variable est modifiée. La complexité de cet algorithme est $O(rb)$.

3.4.2 Filtrage tardif

Puisqu'on souhaite uniquement filtrer la variable y_{ij} qui vient d'être sélectionnée, l'algorithme peut être simplifié. Si une variable à la deuxième ligne ou plus bas est sélectionnée, alors il faut vérifier si elle est dans la première colonne ou non. Si elle est dans la première colonne, alors nous filtrons simplement son domaine dans le but de retirer les valeurs plus petites à la valeur de la variable juste au-dessus. Sinon, alors il faut imposer l'ordre lexicographique tant et aussi longtemps que les valeurs des variables à gauche de y_{ij} sont égales aux valeurs de la variable juste au-dessus (ligne $i-1$). Tout dépendant si y_{ij} est à la dernière colonne ($j = r$) ou non, la manière de filtrer son domaine change légèrement. Cette différence s'explique par le fait que lorsqu'on arrive à la dernière colonne, il ne reste plus aucune chance de pouvoir forcer l'ordre lexicographique. Voir l'algorithme 1 pour le pseudo-code. La fonction Valeur retourne la valeur instanciée d'une variable et la fonction Dom retourne le domaine d'une variable. La complexité de cet algorithme est $O(r + b)$.

Algorithme 1 : LexLigneTardif

```
Entrée : La variable y_ij qui vient d'être sélectionnée
si i > 1 alors
    si j > 1 alors
        si Valeur(y_(i-1),l) = Valeur(y_il)   ∀l ∈ [1..(j − 1)]
        alors
            si j ≠ r alors
                Dom(y_ij) =
                  Dom(y_ij) \ {x | x < Valeur(y_(i-1),j)}
            sinon
                Dom(y_ij) =
                  Dom(y_ij) \ {x | x ≤ Valeur(y_(i-1),j)}
    sinon
        // La première colonne de la matrice doit
           être en ordre non décroissant
        Dom(y_ij) = Dom(y_ij) \ {x | x < Valeur(y_(i-1),j)}
```

3.5 Ordre lexicographique entre les colonnes

3.5.1 Filtrage traditionnel

Pour chaque variable qui a été instanciée dans la matrice, nous devons vérifier dans quelles lignes chaque valeur a été utilisée. Pour les valeurs qui ont été utilisées sur les mêmes lignes, nous devons seulement conserver la plus petite valeur dans le domaine de la variable suivant la variable qui vient d'être instanciée. La complexité de cet algorithme est $O(v(r + b^2))$.

3.5.2 Filtrage tardif

Pour chaque variable qui a été instanciée, nous devons vérifier dans quelles lignes chaque valeur a été utilisée. Pour les valeurs qui ont été instanciées sur les mêmes lignes, nous devons conserver uniquement la valeur la plus petite dans le domaine de la variable sélectionnée. La complexité de cet algorithme est $O(v(r + b^2))$.

4 Expérimentations

Dans cette section, nous comparons le filtrage selon qu'il réponde au filtrage classique ou tardif. Nous procédons d'abord à l'évaluation pour les BIBDs. Ensuite, à l'aide d'un contre-exemple (remplissage d'une matrice auxquelles seule la contrainte LEX est appliquée), nous montrons que le filtrage tardif n'est évidemment pas approprié pour tous les problèmes.

4.1 BIBDs

Nous utilisons des instances qui proviennent de Flener et coll. [1] ainsi que Yokoya et Yamada [18]. Les algorithmes ont été implémentés en C# étant donné que le filtrage tardif était difficile à implémenter dans un solveur générique.

Le serveur est un i5-2.60 GHz avec 8 Gb de mémoire vive.

Pour rappel, les variables sont instanciées en ordre lexicographique (gauche à droite, haut à bas). L'heuristique de choix de valeur sélectionne toujours la plus petite valeur possible en premier. Une stratégie de retour-arrière chronologique (DFS) est appliquée en cas d'échec. Le Tableau 2 montre le temps requis pour explorer complètement les arbres de recherche. Nous présentons aussi le nombre d'échecs ainsi que le nombre de solutions trouvées pour chaque instance BIBD. Comme nous pouvons le voir, le nombre d'échecs est exactement le même selon qu'on utilise une approche ou une autre. Par contre, le temps de résolution se retrouve à être réduit de 20 % à 30 % lorsque le filtrage tardif est utilisé.

Dans le but de mieux analyser les résultats présentés au Tableau 2, l'image 3 montre le temps requis pour explorer jusqu'à un certain pourcentage de l'arbre de recherche pour les deux approches. La relation entre les deux approches est linéaire parce que nous ne changeons pas la complexité des algorithmes ou le niveau de filtrage. Nous montrons ces résultats uniquement pour l'instance $(8, 4, 3)$ dans le but d'économiser de l'espace, mais les résultats sont très similaires lorsqu'on analyse les autres instances.

Nous avons également cherché à déterminer si le gain est distribué uniformément sur l'ensemble des

Tableau 2 – Résultats selon qu'on filtre traditionnellement ou tardivement

Instance	Type algo	Temps (s)	#Échecs	#sols
(6, 3, 2)	Filtrage traditionnel	0,097	486	1
	Filtrage tardif	0,076	486	1
(7, 3, 2)	Filtrage traditionnel	2,227	11 817	12
	Filtrage tardif	1,579	11 817	12
(9, 3, 1)	Filtrage traditionnel	1,077	6 439	2
	Filtrage tardif	0,676	6 439	2
(8, 4, 3)	Filtrage traditionnel	142,989	739 695	92
	Filtrage tardif	102,949	739 695	92
(6, 3, 4)	Filtrage traditionnel	27,280	79 994	21
	Filtrage tardif	21,625	79 994	21
(11, 5, 2)	Filtrage traditionnel	19,511	140 530	2
	Filtrage tardif	13,153	140 530	2
(7, 3, 3)	Filtrage traditionnel	349,753	941 904	220
	Filtrage tardif	266,021	941 904	220

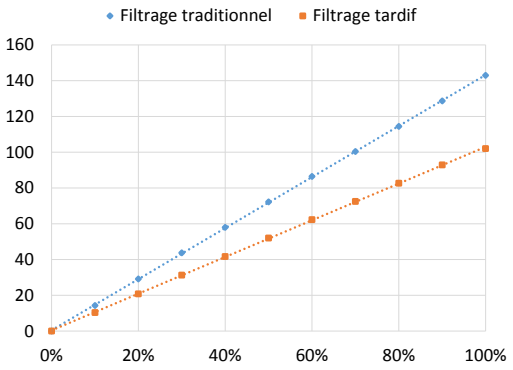


Image 3 – Temps requis en secondes à un certain % d'exploration pour l'instance $(8, 4, 3)$

contraintes. Le Tableau 3 montre comment le temps de calcul est réduit pour chaque contrainte. Nous nous serions attendus à ce qu'il puisse être préférable de filtrer au plus tôt certaines contraintes et d'autres le plus tard possible, mais ce n'est pas le cas pour le problème étudié. La différence est positive pour chaque contrainte. Par contre, l'amélioration diffère grandement d'une contrainte à une autre.

4.2 Contre-exemple : conserver seulement la contrainte d'ordre lexicographique

À titre de contre-exemple, nous avons défini un problème qui consiste à remplir une matrice et de lui appliquer la contrainte d'ordre lexicographique strict (LEX) entre les lignes. Nous effectuons le filtrage soit en version traditionnelle (Frisch et coll.) ou tardive (voir 3.4.2).

Plus précisément, nous considérons des tableaux de variables de 2 lignes et c colonnes où, selon l'instance, $c \in \{5, 6, 7\}$. Chaque variable a pour domaine l'ensemble {1, 2, 3}. Il y a une contrainte LEX entre les deux lignes.

Nous voyons que l'ordre lexicographique filtré de manière traditionnelle (Frisch et coll.) et le filtrage tardif définissent le même espace de solutions, mais que le filtrage est plus efficace avec le filtrage traditionnel (moins d'échecs et donc moins de retours-arrière).

5 Conclusion

Nous souhaitions savoir quel était le meilleur moment pour exécuter les algorithmes de filtrage lorsqu'on tente de résoudre un BIBD. Nous montrons qu'en retardant le filtrage jusqu'au dernier moment (filtrer le domaine d'une variable uniquement lorsque la variable vient d'être sélectionnée et sur le point d'être instanciée) réduit le temps de calcul entre 20 % et 30 % pour les instances présentées. L'approche a préservé le même espace solution et le même niveau de filtrage.

Il va de soi que ces résultats sont spécifiques au problème que nous avons étudié. Par contre, il est possible que d'autres genres de problèmes puissent aussi bénéficier d'un filtrage tardif. Pour certains problèmes, l'approche optimale pourrait être un mélange de filtrage classique et tardif. Les solveurs génériques actuels ne donnent pas accès à l'événement *variableAÉté-Sélectionnée*. Bien qu'il ne s'agisse pas d'une panacée, nous pensons qu'ajouter cet événement aux solveurs de PPC pourrait faciliter l'implémentation, le test et finalement l'utilisation d'algorithmes de filtrage tardif pour d'autres problèmes qui pourraient en bénéficier.

Tableau 3 – Gain en temps pour les contraintes de l'instance (8, 4, 3)

	Temps filtrage traditionnel (s)	**Temps filtrage tardif (s)**	**Réduction en %**
Contrainte ordre croissant dans ligne	6,094	2,160	64,56 %
Contrainte k	11,945	2,438	79,59 %
Contrainte λ	23,606	6,247	73,54 %
Ordre lex lignes	1,117	0,117	89,56 %
Ordre lex colonnes	89,667	75,019	16,34 %

Tableau 4 – Frisch et coll. vs filtrage tardif pour ordre lexicographique strict entre 2 lignes

#Colonnes	**Type algo**	**Temps (s)**	**#Échecs**	**#sols**
5	Frisch et coll.	0,296	0	29 403
	Filtrage tardif	0,316	81	29 403
6	Frisch et coll.	2,330	0	265 356
	Filtrage tardif	2,438	243	265 356
7	Frisch et coll.	20,648	0	2 390 391
	Filtrage tardif	21,046	729	2 390 391

Références

[1] Pierre Flener, Alan Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models. In *Principles and Practice of Constraint Programming-CP 2002*, pages 462–477. Springer, 2002.

[2] Pierre Flener, Justin Pearson, and Luis G Reyna. Financial portfolio optimisation. In *International Conference on Principles and Practice of Constraint Programming*, pages 227–241. Springer, 2004.

[3] Alan Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Global constraints for lexicographic orderings. In *Principles and Practice of Constraint Programming-CP 2002*, pages 93–108. Springer, 2002.

[4] Alan Frisch, Christopher Jefferson, and Ian Miguel. Symmetry breaking as a prelude to implied constraints : A constraint modelling pattern. In *ECAI*, volume 16, page 171, 2004.

[5] Gecode Team. Gecode : Generic constraint development environment, 2006. Disponible sur `http://www.gecode.org`.

[6] Takakazu Kurokawa and Yoshiyasu Takefuji. Neural network parallel computing for bibd problems. *IEEE Transactions on Circuits and Systems II : Analog and Digital Signal Processing*, 39(4) :243–247, 1992.

[7] François Pachet and Pierre Roy. Automatic generation of music programs. In *International Conference on Principles and Practice of Constraint Programming*, pages 331–345. Springer, 1999.

[8] Steven Prestwich. Balanced incomplete block design as satisfiability. In *Proceedings of the 12th Irish Conference on Artificial Intelligence and Cognitive Science*, 2001.

[9] Steven Prestwich. *A local search algorithm for balanced incomplete block designs*, pages 132–143. Springer, 2003.

[10] Patrick Prosser and Evgeny Selensky. *A study of encodings of constraint satisfaction problems with 0/1 variables*, pages 121–131. Springer, 2003.

[11] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. Disponible sur `http://www.choco-solver.org`.

[12] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming.* Elsevier, 2006.

[13] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the thirteenth national conference on Artificial intelligence-Volume 1*, pages 209–215. AAAI Press, 1996.

[14] Christian Schulte and Peter J Stuckey. When do bounds and domain propagation lead to the

same search space ? *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 27(3) :388–425, 2005.

[15] Christian Schulte and Peter J Stuckey. Efficient constraint propagation engines. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31(1) :2, 2008.

[16] Christian Schulte and Guido Tack. Implementing efficient propagation control. *TRICS*, 2010.

[17] Douglas R. Stinson. *Combinatorial Designs : Constructions and Analysis*. Springer, New York, 2004.

[18] Daisuke Yokoya and Takeo Yamada. A mathematical programming approach to the construction of bibds. *International Journal of Computer Mathematics*, 88(5) :1067–1082, 2011.

Autoriser des tuples interdits pour rendre une instance CSP traitable

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

Aix Marseille Univ, Université de Toulon, CNRS, ENSAM, LSIS, Marseille, France
{achref.elmouelhi, philippe.jegou, cyril.terrioux}@lsis.org

Résumé

Les classes polynomiales pour lesquelles l'arc-cohérence est une procédure de décision ont souvent intéressé les chercheurs de la communauté CP, et cela, pour différentes raisons. Parmi celles-ci, nous pouvons citer la faible complexité en temps du filtrage par cohérence d'arc et la conservation de la structure de l'instance CSP, ce qui permet souvent de préserver les propriétés à l'origine de la polynomialité. Des efforts considérables ont été accomplis pour étendre ces classes polynomiales. Dans ce contexte, des travaux sur les décompositions arborescentes et les graphes triangulés ont été exploités par la communauté CP. Par exemple, il a été montré que la cohérence d'arc est une procédure de décision pour les instances CSP dont la microstructure est triangulée. Pour autant, ces efforts n'ont pas été pleinement récompensés dans le cas de la propriété BTP (Broken-Triangle Property). Plus précisément, hormis WBTP et m-fBTP, la plupart des extensions de BTP, comme k-BTP ou m-wBTP, requiert un niveau de cohérence plus élevé, ce qui les rend inutilisables au-delà de $k = 3$ et $m = 2$.

Aussi, dans ce papier, nous identifions un nouvel outil théorique permettant de transformer les instances en instances équivalentes qui satisfont la propriété BTP, et cela, en autorisant certains tuples interdits. Cette opération, qui peut être effectuée en temps polynomial, est appelée BTPisation. Nous étudions ses propriétés et nous la comparons à des classes polynomiales existantes basées sur la microstructure des instances CSP.

Abstract

Tractable classes for which arc-consistency is a decision procedure have attracted researchers in Constraint Programming (CP) for several reasons. Among them, we cite the low complexity of the arc-consistency filtering and the conservation of the CSP structure, which often permits not to loose tractable properties. Considerable research efforts have been made in order to extend these tractable classes. In this direction, works on tree decomposition and triangulated graphs have been re-used in the CP community. For example, it has been shown that arc-consistency is a decision procedure for CSP instances whose microstructure is triangulated. These efforts have not been fully rewarded in the case of BTP (Broken-Triangle Property). More precisely, except WBTP and m-fBTP, most extensions of BTP like k-BTP and m-wBTP, need a high level of consistency, which make them unusable beyond $k = 3$ and $m = 2$.

So, in this paper, we identify a new theoretical tool allowing to transform instances in equivalent instances satisfying BTP, by authorizing some specific forbidden tuples. This operation which can be achieved in polynomial-time, is called BTPization. We study its properties while comparing it to existing tractable classes founded on the micro-structure of CSPs.

1 Introduction

Dans le cadre de l'étude théorique du problème de satisfaction de contraintes, un axe de recherche important porte sur l'identification de classes polynomiales. Récemment, différents travaux ont été menés pour étendre des classes existantes, soit en allégeant les propriétés requises [15, 1, 12, 6, 16, 3, 7], qui sont souvent trop restrictives, soit en appliquant certaines transformations comme les opérations de filtrages [8].

Dans ce papier, nous introduisons une nouvelle approche qui transforme certaines instances CSP binaires, qui n'appartiennent pas initialement à une classe polynomiale connue, en instances appartenant à une telle classe. Si cette transformation s'effectue dans le même esprit que celle introduite dans [8], elle s'en distingue significativement. Notamment, elle ne satisfait pas la définition de transformation proposée dans [8]. Plus précisément, étant donnée une instance binaire qui ne satisfait pas la propriété BTP (Broken-Triangles Property [5]), nous montrons comment nous pouvons autoriser certains tuples interdits pour élimi-

ner certains triangles cassés tout en préservant l'équivalence en termes d'ensemble de solutions et en obtenant ainsi des instances qui sont désormais résolubles en temps polynomial. Ces tuples sont ajoutés par une transformation polynomiale que nous appelons *BTPisation*.

Dans ce but, nous introduisons une nouvelle classe polynomiale, appelée m-fRBTP pour fully Reparable Broken-Triangle Property (qui ne doit pas être confondue avec la classe m-fBTP introduite récemment dans [7]) qui généralise BTP. Résoudre des instances binaires satisfaisant m-fRBTP peut être réalisé en autorisant d'abord certains tuples interdits, afin d'éliminer certains triangles cassés, puis en appliquant la cohérence d'arc [14]. Nous comparons cette nouvelle extension de BTP à d'autres comme BTP^{AC}[8], k-BTP [6] et m-fBTP [7], établissant ainsi des relations d'incomparabilité ou d'inclusion entre elles. Nous montrons aussi que la présence de certains triangles cassés sur un couple de valeurs d'une instance binaire n'empêche pas de les fusionner sans affecter la satisfiabilité de l'instance si ce couple satisfait m-fRBTP.

L'article est organisé ainsi. D'abord, nous rappelons le contexte formel dans la section 2 avant de définir l'opération de BTPisation dans la section 3. Ensuite, nous introduisons les propriétés de triangles cassés (partiellement, pleinement) réparables dans la section 4. Puis, la section 5 présente les liens entre cette nouvelle classe polynomiale et certaines des classes existantes basées sur BTP. Dans la section 6, nous montrons que m-fRBTP autorise la fusion de valeurs mais pas l'élimination de variables. Enfin, nous concluons dans la section 7.

2 Contexte

Nous commençons par rappeler la définition d'une instance binaire du problème de satisfaction de contraintes (Constraint Satisfaction Problem (CSP)) :

Définition 1 (Instance CSP) *Une instance CSP binaire $I = (X, D, R)$ consiste en :*

- *un ensemble $X = \{x_1, ..., x_n\}$ de n **variables**,*
- *un ensemble $D = \{D(x_1), ..., D(x_n)\}$ de **domaines** finis de **valeurs**, à raison d'un domaine par variable,*
- *un ensemble R de **relations** binaires. Pour chaque paire de variables (x_i, x_j), il existe une relation $R_{ij} \subseteq D(x_i) \times D(x_j)$ qui définit l'ensemble des combinaisons de valeurs compatibles, appelés **tuples autorisés** pour (x_i, x_j).*

Une affectation $(v_{\ell_1}, \ldots, v_{\ell_m}) \in D(x_{\ell_1}) \times \ldots \times D(x_{\ell_m})$ est dite *cohérente* si elle satisfait toutes les relations R_{ij} telles que $\{x_i, x_j\} \subseteq \{x_{\ell_1}, \ldots, x_{\ell_m}\}$ (c'est-à-dire $(v_i, v_j) \in R_{ij}$). Une *solution* est une affectation cohérente de toutes les variables de X. Déterminer si une instance CSP binaire I possède une solution est un problème connu pour être NP-complet. Cependant, en imposant certaines restrictions, la résolution peut être accomplie en temps polynomial. Par exemple, c'est le cas pour les instances satisfaisant la propriété BTP. Cette propriété, qui repose sur l'absence de *triangle cassé*, se définit formellement ainsi :

Définition 2 (Broken-Triangle Property [5]) *Soit I une instance CSP binaire dotée d'un ordre $<$ sur les variables. Une paire de valeurs $v'_k, v''_k \in D(x_k)$ satisfait la propriété BTP si, pour chaque paire de variables (x_i, x_j), $\forall v_i \in D(x_i)$, $\forall v_j \in D(x_j)$, si $(v_i, v_j) \in R_{ij}$, $(v_i, v'_k) \in R_{ik}$ et $(v_j, v''_k) \in R_{jk}$, alors $(v_i, v''_k) \in R_{ik}$ ou $(v_j, v'_k) \in R_{jk}$.*

Une variable x_k satisfait la propriété BTP si chaque paire de valeurs de $D(x_k)$ satisfait BTP. L'instance I satisfait la propriété BTP par rapport à l'ordre $<$ sur les variables si, pour chaque variable x_k, x_k satisfait BTP dans la sous-instance de I restreinte aux variables x_i telles que $x_i \leq x_k$.

Si $(v_i, v_j) \in R_{ij}$, $(v_i, v'_k) \in R_{ik}$, $(v_j, v''_k) \in R_{jk}$ $(v_i, v''_k) \notin R_{ik}$ et $(v_j, v'_k) \notin R_{jk}$, alors le quadruplet (v_i, v_j, v'_k, v''_k) constitue un *triangle cassé* pour les valeurs v'_k et v''_k (ou plus généralement pour x_k) vis-à-vis de x_i et x_j. Dans ce cas-là, on dit également que x_i et x_j contribuent à l'existence d'un triangle cassé pour les valeurs v'_k et v''_k.

Graphiquement, cette propriété peut être visualisée dans le *graphe de microstructure*[1] [11], comme le montre la figure 1. Notons que, dans l'ensemble du papier, pour chaque paire de valeurs (v_i, v_j) (avec $v_i \in D(x_i)$, $v_j \in D(x_j)$ et $x_i \neq x_j$), v_i et v_j sont liés par une arête en trait plein si (v_i, v_j) est un tuple autorisé (c'est-à-dire $(v_i, v_j) \in R_{ij}$), ou une arête en pointillés si (v_i, v_j) est un tuple interdit (c'est-à-dire $(v_i, v_j) \notin R_{ij}$). L'absence d'arête entre v_i et v_j signifie alors que (v_i, v_j) est un tuple indéfini qui peut correspondre soit à un tuple autorisé, soit à un tuple interdit. Si nous considérons l'ordre sur les variables $x_i < x_j < x_k$, l'instance CSP de la figure 1(a) ne satisfait pas la propriété BTP selon cet ordre puisque qu'on a $(v_j, v'_k) \notin R_{ik}$ et $(v_i, v''_k) \notin R_{jk}$. Ainsi, le quadruplet (v_i, v_j, v'_k, v''_k) constitue un triangle cassé. En revanche, dans la figure 1(b), comme on a $(v_i, v''_k) \in R_{ik}$ et $(v_j, v'_k) \in R_{jk}$, la propriété BTP selon cet ordre est satisfaite.

1. Étant donnée une instance CSP binaire $I = (X, D, R)$, le *graphe de microstructure* (aussi appelée la *microstructure*) de I est le graphe non-orienté $\mu(I) = (V, E)$ avec $V = \{(x_i, v_i) \mid x_i \in X, v_i \in D(x_i)\}$ et $E = \{ \{(x_i, v_i), (x_j, v_j)\} \mid i \neq j, (v_i, v_j) \in R_{ij})\}$.

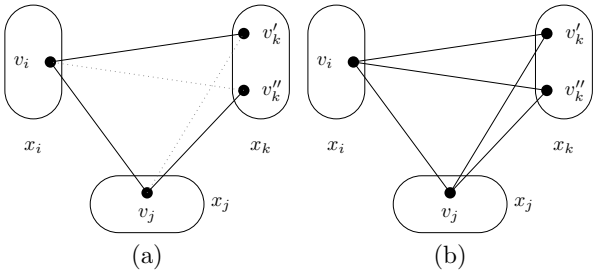


FIGURE 1 – Le quadruplet (v_i, v_j, v'_k, v''_k) forme un triangle cassé dans (a) mais pas dans (b) grâce aux arêtes $\{(x_i, v_i), (x_k, v''_k)\}$ et $\{(x_k, v_j), (x_k, v'_k)\}$.

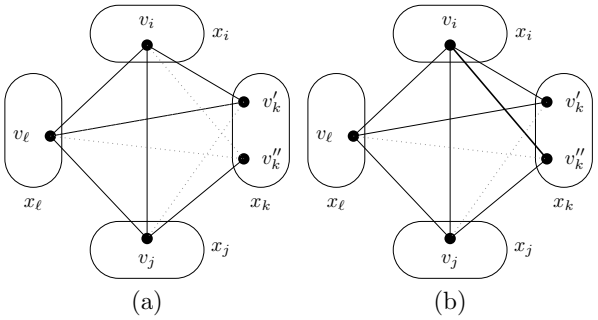


FIGURE 2 – L'élimination de n'importe quel triangle cassé empêche l'élimination de l'autre.

En pratique, les instances satisfaisant cette propriété sont relativement rares, comme cela est souvent le cas pour la plupart des autres classes polynomiales, du fait de la présence de triangles cassés. Dans [8], il a été montré que certains triangles cassés pouvaient être éliminés en appliquant un certain niveau de cohérence. Ici, nous allons établir que d'autres triangles cassés peuvent être supprimés en autorisant certains tuples qui sont initialement interdits, sans pour autant impacter l'ensemble des solutions de l'instance. L'ajout de tels tuples peut alors, dans certains cas, permettre de rendre l'instance BTP.

3 BTPisation

Autoriser des tuples interdits, afin d'éliminer des triangles cassés, peut induire de nouveaux triangles cassés ou rendre impossible l'ajout de tout autre tuple interdit, empêchant ainsi l'élimination d'autres triangles cassés.

La microstructure présentée dans la figure 2(a) contient deux triangles cassés : (v_i, v_j, v'_k, v''_k) et $(v_j, v_\ell, v''_k, v'_k)$. Éliminer le premier en autorisant le tuple (v_i, v''_k) empêche ensuite l'élimination du second. Autrement dit, une fois ajouté (v_i, v''_k) à R_{ik}, nous ne

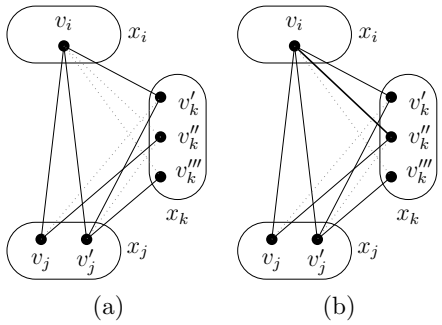


FIGURE 3 – Un nouveau triangle cassé $(v_i, v_j, v''_k, v'''_k)$ apparaît quand le triangle cassé (v_i, v_j, v'_k, v''_k) est éliminé en autorisant (v_i, v''_k).

pouvons plus ajouter (v_ℓ, v''_k) à $R_{\ell k}$, ni (v_j, v'_k) à R_{jk} car cela remettrait en cause la satisfiabilité de l'instance. Il en est de même si nous commençons par autoriser (v_j, v'_k).

Dans la microstructure de la figure 3(a), il existe un unique triangle cassé, à savoir (v_i, v_j, v'_k, v''_k). L'ajout de l'arête (v_i, v''_k) pour le supprimer (voir la figure 3(b)) a pour conséquence la création d'un nouveau triangle cassé, à savoir $(v_i, v'_j, v''_k, v'''_k)$, qui n'existait pas auparavant.

Ainsi, de nombreuses règles peuvent être définies pour éliminer des triangles cassés, mais, ici, nous nous intéressons uniquement à l'opération de BTPisation.

Définition 3 *Étant donnée une instance CSP binaire I qui ne satisfait pas la propriété BTP, une opération de* BTPisation *consiste à autoriser certains tuples initialement interdits dans I afin d'obtenir une nouvelle instance I' satisfaisant BTP tout en préservant l'équivalence entre I et I' (c'est-à-dire que I et I' ont le même ensemble de solutions) et sans introduire de nouveaux triangles cassés.*

4 Les triangles cassés (partiellement, pleinement) réparables

Comme nous venons de l'évoquer dans la section 3, autoriser certains tuples interdits peut éliminer certains triangles cassés. Dans la figure 4(a), nous pouvons remarquer que le quadruplet (v_i, v_j, v'_k, v''_k) forme un triangle cassé. Par conséquent, l'instance initiale ne satisfait pas la propriété si on considère que $x_k > max(x_i, x_j)$. Après l'ajout du tuple (v_i, v''_k) à R_{ik} (voir la figure 4(b)) ou du tuple (v_j, v'_k) à R_{jk}, l'instance obtenue appartient à la classe BTP.

Grâce à la figure 4, nous pouvons voir que certains triangles cassés sont *réparables*. Dans la figure 1(a), ajouter (v_i, v''_k) à R_{ik} ou (v_j, v'_k) à R_{jk} remet en cause

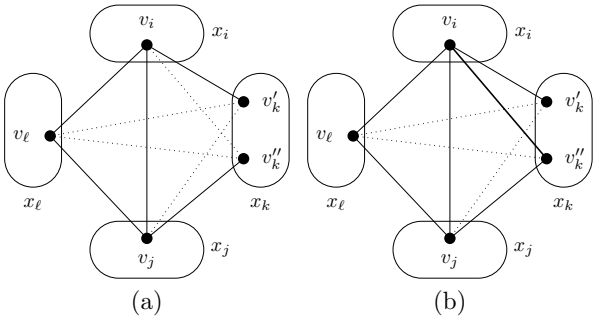


FIGURE 4 – Une instance CSP I qui ne satisfait pas la propriété BTP selon l'ordre $x_i < x_j < x_k$. I devient BTP après l'autorisation du tuple (v_i, v''_k).

la satisfiabilité de l'instance. Pour éviter ce problème, nous allons exploiter des variables supports comme x_ℓ dans la figure 4. Ici, x_ℓ est une variable support dans le sens où elle a un impact non négligeable au niveau de la conservation de l'équivalence. En effet, cette variable support permet d'empêcher l'introduction d'une nouvelle solution lors de l'ajout d'un tuple (v_i, v''_k). Ce concept de *variables supports* a été précédemment introduit dans [3]. Cependant, l'usage que nous en ferons ici nécessite une légère modification de la définition originale. Les trois nouvelles propriétés reposent sur l'emploi de m variables supports.

4.1 Les triangles cassés partiellement réparables

Nous commençons par la première propriété que nous appellerons m-pRBTP (pour *partially Reparable Broken-Triangle Property*).

Notation 1 *Étant donnée une variable $x_k \in X$, $\neg BT(x_k)$ représente l'ensemble des variables qui ne contribuent pas à un triangle cassé sur x_k.*

Définition 4 *Étant donné un triangle cassé (v_i, v_j, v'_k, v''_k) avec $v_i \in D(x_i)$, $v_j \in D(x_j)$ et $v'_k, v''_k \in D(x_k)$, un ensemble $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \subseteq \neg BT(x_k)$ supporte (v_i, v_j, v'_k, v''_k) si pour toute affectation cohérente $(v_{\ell_1}, \ldots, v_{\ell_r}, v_i)$ avec $\forall t \in \{1, \ldots, r\}, v_{\ell_t} \in D(x_{\ell_t})$, il existe $\alpha \in \{1, \ldots, r\}$ tel que $(v'_k, v_{\ell_\alpha}) \notin R_{k\ell_\alpha}$ ou $(v''_k, v_{\ell_\alpha}) \notin R_{k\ell_\alpha}$.*

La propriété m-pRBTP repose sur l'existence, pour chaque variable x_k, d'un ensemble support commun à tous les triangles cassés sur x_k.

Définition 5 *Une variable x_k satisfait la propriété m-**pRBTP** avec $m \leq n-3$ s'il existe un ensemble $S(x_k)$ de $r \leq m$ variables $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \subseteq \neg BT(x_k)$ qui supporte chaque triangle cassé sur x_k.*

*Dans ce cas, les triangles cassés sur x_k sont dits partiellement m-**réparables**.*
*$x_{\ell_1}, \ldots, x_{\ell_r}$ sont appelées **variables supports** de x_k.*

Dans la suite, on note par $\mathcal{S}_k$ l'ensemble de tous les ensembles $S(x_k)$ qui supportent tous les triangles cassés sur x_k. En l'absence d'ambiguïté, nous écrirons réparable pour m-réparable.

La figure 5 présente deux configurations différentes d'instances qui satisfont la propriété 2-pRBTP. Dans la première microstructure, la variable x_k satisfait la propriété 2-pRBTP car, pour les deux affectations cohérentes $(v'_{\ell_\beta}, v'_{\ell_\gamma}, v_i)$ et $(v''_{\ell_\beta}, v''_{\ell_\gamma}, v_i)$, nous avons $(v'_{\ell_\gamma}, v'_k), (v'_{\ell_\gamma}, v''_k) \notin R_{\ell_\gamma k}$ et $(v''_{\ell_\gamma}, v'_k), (v''_{\ell_\gamma}, v''_k) \notin R_{\ell_\gamma k}$. Dans la seconde microstructure, la variable x_k satisfait aussi la propriété 2-pRBTP puisque, pour les deux affectations cohérentes $(v'_{\ell_\beta}, v'_{\ell_\gamma}, v_i)$ et $(v''_{\ell_\beta}, v''_{\ell_\gamma}, v_i)$, nous avons $(v'_{\ell_\gamma}, v'_k), (v'_{\ell_\gamma}, v''_k) \notin R_{\ell_\gamma k}$ et $(v''_{\ell_\beta}, v'_k), (v''_{\ell_\beta}, v''_k) \notin R_{\ell_\beta k}$.

Dans la figure 5(a), on peut observer que la variable x_k satisfait aussi la propriété 1-pRBTP parce que la variable x_{ℓ_γ} supporte, à elle toute seule, le triangle cassé (v_i, v_j, v'_k, v''_k). Cela nous conduit à formuler le résultat suivant :

Proposition 1 *Pour toute instance binaire ayant n variables, si une variable x_k satisfait la propriété m-pRBTP, alors elle satisfait également la propriété $(m+1)$-pRBTP (pour $0 \leq m \leq n-4$).*

À présent, nous montrons que l'élimination de triangles cassés réparables, en ajoutant un des tuples manquants, ne modifie pas l'ensemble des solutions de l'instances

Proposition 2 *Étant donnés une variable x_k satisfaisant la propriété m-pRBTP et un triangle cassé (v_i, v_j, v'_k, v''_k) avec $v_i \in D(x_i)$, $v_j \in D(x_j)$ et $v'_k, v''_k \in D(x_k)$, le tuple (v_i, v''_k) peut être ajouté à R_{ik} tout en conservant l'équivalence.*

Preuve : Soient I l'instance originale et I' la nouvelle instance produite à partir de I en ajoutant le tuple (v_i, v''_k) à R_{ik} afin de réparer le triangle cassé (v_i, v_j, v'_k, v''_k). Nous pouvons facilement remarquer que si I est satisfiable, alors I' l'est aussi. Aussi, il nous suffit de démontrer que si I' possède une solution s', alors s' doit aussi être une solution de I.

Supposons, en vue d'obtenir une contradiction, que s' ne soit pas une solution de I. Logiquement, si s' est une solution de I' mais pas de I, cela est dû au tuple (v_i, v''_k) nouvellement ajouté à R_{ik}. Par conséquent, nous avons $s'(x_i) = v_i$ et $s'(x_k) = v''_k$ (en notant $s'(x)$ la valeur de la variable x dans s').

Puisque la variable x_k satisfait la propriété m-pRBTP, alors il existe un ensemble $S(x_k)$ de

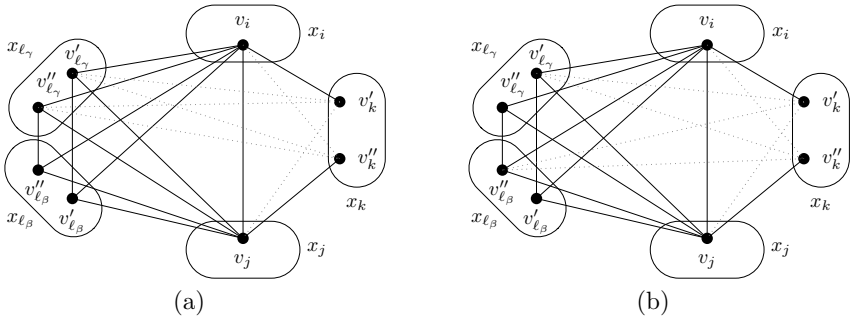


FIGURE 5 – Deux configurations différentes d'instances satisfaisant la propriété 2-pRBTP.

$r \leq m$ variables $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \subseteq \neg BT(x_k)$ tel que pour chaque triangle cassé (v_i, v_j, v'_k, v''_k) avec $v_i \in D(x_i)$, $v_j \in D(x_j)$ et $v'_k, v''_k \in D(x_k)$ et pour toute affectation cohérente $(v_{\ell_1}, \ldots, v_{\ell_r}, v_i)$ sur $S(x_k) \cup \{x_i\}$, il existe $\alpha \in \{1, \ldots, r\}$ tel que $(v'_k, v_{\ell_\alpha}) \notin R_{k\ell_\alpha}$ ou $(v''_k, v_{\ell_\alpha}) \notin R_{k\ell_\alpha}$. Par définition d'une affectation cohérente, nous avons $(s'(x_i), s'(x_{\ell_\alpha})) \in R_{i\ell_\alpha}$ et $(s'(x_k), s'(x_{\ell_\alpha})) \in R_{k\ell_\alpha}$. Puisque $(s'(x_k), s'(x_{\ell_\alpha})) \in R_{k\ell_\alpha}$, nous devons avoir $(v'_k, s'(x_{\ell_\alpha})) \notin R_{k\ell_\alpha}$. Dans ce cas, nous avons un triangle cassé $(v_i, v_{\ell_\alpha}, v'_k, v''_k)$ dans I. Or, par définition, nous avons supposé que $x_{\ell_\alpha} \in \neg BT(x_k)$. Ainsi, cela contredit notre hypothèse. Donc, s' ne peut pas être une solution de I', ni de I. Par conséquent, ajouter le tuple (v_i, v''_k) à R_{ik} ne modifie pas l'ensemble de solution de I. □

Lemme 1 *Étant donnés une variable x_k satisfaisant la propriété m-pRBTP et un triangle cassé (v_i, v_j, v'_k, v''_k), si l'ajout du tuple (v_i, v''_k) à R_{ik} introduit de nouveaux triangles cassés sur x_k vis-à-vis de x_i et x_j, alors il existe au moins un $S(x_k) \in \mathcal{S}_k$ qui les supporte.*

Preuve : Considérons une variable x_k satisfaisant la propriété m-pRBTP et un triangle cassé (v_i, v_j, v'_k, v''_k). Supposons, en vue d'obtenir une contradiction, que l'ajout du tuple (v_i, v''_k) à R_{ik} engendre de nouveaux triangles cassés sur x_k vis-à-vis de x_i et x_j tels qu'il n'existe aucun $S'(x_k) \in \mathcal{S}_k$ qui les supporte. Soit $S(x_k) \in \mathcal{S}_k$. Forcément, il existe un nouveau triangle cassé $(v_i, v'_j, v''_k, v'''_k)$ qui n'est pas supporté par $S(x_k)$. Nous avons nécessairement $v'''_k \neq v'_k$ et $v'_j \neq v_j$.

Comme $(v_i, v'_j, v''_k, v'''_k)$ n'est pas supporté par $S(x_k)$, il existe une affectation cohérente $\mathcal{A} = (v_{\ell_1}, \ldots, v_{\ell_r}, v_i)$ telle que, pour chaque variable $x_\ell \in S(x_k)$, nous avons :

- $(v''_k, \mathcal{A}(x_\ell)) \in R_{k\ell}$ et
- $(v'''_k, \mathcal{A}(x_\ell)) \in R_{k\ell}$.

Deux cas sont possibles :

1. $(v'_k, \mathcal{A}(x_\ell)) \notin R_{k\ell}$. C'est impossible car $(v_i, \mathcal{A}(x_\ell), v'_k, v''_k)$ serait un triangle cassé et que nous avons supposé que $x_\ell \in \neg BT(x_k)$.
2. $(v'_k, \mathcal{A}(x_\ell)) \in R_{k\ell}$. Dans ce cas, (v_i, v_j, v'_k, v''_k) ne serait pas supporté par $S(x_k)$ car $(v_i, \mathcal{A}(x_\ell)) \in R_{i\ell}$, $(v'_k, \mathcal{A}(x_\ell)) \in R_{k\ell}$ et $(v''_k, \mathcal{A}(x_\ell)) \in R_{k\ell}$. D'où une contradiction.

En conséquence, les triangles cassés potentiellement introduits par l'ajout du tuple (v_i, v''_k) à R_{ik} sont forcément supportés par un ensemble de $\mathcal{S}_k$. □

Malheureusement, le lemme 1 ne peut être généralisé à n'importe quel triplet de variables (x_i, x_p, x_k) avec $p \neq j$, c'est-à-dire qu'ajouter (v_i, v''_k) à R_{ik} peut créer un triangle cassé sur x_k vis-à-vis de x_i et de toute autre variable de $X \setminus \{x_i, x_k, x_{\ell_1}, \ldots, x_{\ell_r}\}$ qui ne soit pas supporté par un élément de $\mathcal{S}_k$. Pour cela, nous introduisons les *triangles cassés réparables*.

4.2 Les triangles cassés réparables

Nous énonçons la définition de la propriété m-RBTP (pour *Reparable Broken-Triangle Property*).

Définition 6 *Une variable x_k satisfait la propriété m-**RBTP** avec $m \leq n - 3$ si :*

1. *x_k satisfait m-pRBTP et*
2. *il existe au moins un ensemble $S(x_k) \in \mathcal{S}_k$ tel que, pour tout triangle cassé (v_i, v_j, v'_k, v''_k) sur x_k, pour toute affectation cohérente $\mathcal{A} = (v_{\ell_1}, \ldots, v_{\ell_r}, v_i)$ de $S(x_k) \cup \{x_i\}$ et pour tout $v_\ell \in \mathcal{A}$, $(v_j, v_\ell) \in R_{j\ell}$ et pour toute variable $x_t \notin S(x_k)$, pour tout $v_t \in D(x_t)$, si $(v_i, v_t) \in R_{it}$ alors $(v'_k, v_t) \in R_{kt}$ et $(v''_k, v_t) \in R_{kt}$.*

Nous pouvons maintenant généraliser le lemme 1 à tout triplet de variables (x_i, x_p, x_k) avec $p \neq j$.

Lemme 2 *Étant donnés une variable x_k satisfaisant la propriété m-RBTP et un triangle cassé*

(v_i, v_j, v'_k, v''_k), *ajouter le tuple* (v_i, v''_k) *à* R_{ik} *n'introduit pas de nouveau triangle cassé sur* x_k *vis-à-vis de* x_i *et* $x_p (\neq x_j)$.

Preuve : Considérons une variable x_k satisfaisant la propriété m-RBTP et un triangle cassé (v_i, v_j, v'_k, v''_k). Supposons, en vue d'obtenir une contradiction, que l'ajout du tuple (v_i, v''_k) à R_{ik} engendre un nouveau triangle cassé sur x_k vis-à-vis de x_i et une variable $x_p (\neq x_j)$. Soit $(v_i, v_p, v''_k, v'''_k)$ ce triangle.

Comme x_k satisfait la propriété m-RBTP, elle satisfait m-pRBTP et il existe un ensemble $S(x_k)$ de $r \leq m$ variables $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \subseteq \neg BT(x_k)$ qui supporte chaque triangle cassé sur x_k. En plus, pour tout triangle cassé (v_i, v_j, v'_k, v''_k) sur x_k, pour toute affectation cohérente $\mathcal{A} = (v_{\ell_1}, \ldots, v_{\ell_r}, v_i)$ de $S(x_k) \cup \{x_i\}$ et pour tout $v_\ell \in \mathcal{A}$, $(v_j, v_\ell) \in R_{j\ell}$ et pour toute variable $x_t \notin S(x_k)$, pour tout $v_t \in D(x_t)$, si $(v_i, v_t) \in R_{it}$ alors $(v'_k, v_t) \in R_{kt}$ et $(v''_k, v_t) \in R_{kt}$.

Nous avons forcément $x_p \in S(x_k)$. En effet, si $x_p \notin S(x_k)$, comme $(v_i, v_p) \in R_{ip}$, nous avons $(v''_k, v_p) \in R_{kp}$, ce qui est incompatible avec le fait que $(v_i, v_p, v''_k, v'''_k)$ est un triangle cassé.

Comme $x_p \in S(x_k)$, on a $(v_i, v_p) \in R_{ip}$ et $(v_j, v_p) \in R_{jp}$. Deux cas sont possibles :

- $(v'_k, v_p) \notin R_{kp}$. Ainsi, le quadruplet $(v_i, v_p, v''_k, v'''_k)$ formerait un triangle cassé. Or, on a supposé que $x_p \in S(x_k)$ qui est inclus dans $\neg BT(x_k)$. Donc ceci est impossible.
- $(v'_k, v_p) \in R_{kp}$. Alors, le quadruplet (v_j, v_p, v''_k, v'_k) formerait aussi un triangle cassé. Or, on a supposé que $x_p \in S(x_k)$ qui est inclus dans $\neg BT(x_k)$. Donc ceci est impossible.

Par conséquent, x_p ne peut être dans $S(x_k)$.

Ainsi, quelle que soit la variable x_p, ajouter le tuple (v_i, v''_k) à R_{ik} ne peut introduire de triangle cassé sur x_k. □

Nous donnons maintenant la définition des instances satisfaisant la propriété m-RBTP.

Définition 7 *Une instance binaire I dotée d'un ordre $<$ sur les variables satisfait la propriété m-RBTP ($m \leq n-3$) par rapport à l'ordre $<$ si, pour toute variable x_k, x_k satisfait la propriété m-RBTP dans la sous-instance de I restreintes aux variables x_i telles que $x_i \leq x_k$.*

On peut facilement en déduire que 0-RBTP et 0-pRBTP correspondent à BTP puisqu'aucune variable support n'est nécessaire du fait de l'absence de tout triangle cassé.

Étant donnée une instance binaire $I = (X, D, R)$ satisfaisant la propriété m-RBTP, cette propriété n'est pas remise en cause vis-à-vis de toute variable $x_i \leq x_k$ suite à l'ajout de (v_i, v''_k) à R_{ik} pour éliminer le triangle cassé (v_i, v_j, v'_k, v''_k) sur x_k. Par contre, elle peut l'être pour une variable $x_p > x_k$, comme le montre la figure 6. Pour éviter cela, nous introduisons, dans la section suivante, la notion de triangle cassé pleinement réparable.

4.3 Les triangles cassés pleinement réparables

Afin de garantir qu'autoriser certains tuples initialement interdits ne crée pas de nouveaux triangles cassés irréparables, nous définissons la notion de triangles cassés bloquants.

Définition 8 *Un triangle cassé m-réparable (v_i, v_j, v'_k, v''_k) avec $v_i \in D(x_i)$, $v_j \in D(x_j)$ et $v'_k, v''_k \in D(x_k)$ est dit m-**bloquant** par rapport à une variable $x_p \in X \setminus \{x_i, x_j, x_k\}$, si les triangles cassés sur x_p créés par l'ajout de (v_i, v''_k) dans R_{ik} ne sont supportés par aucun élément de $\mathcal{S}_p$.*

x_k est dite m-bloquante *par rapport à x_p s'il existe au moins un triangle cassé m-bloquant sur x_k vis-à-vis de x_p.*

Le triangle cassé réparable de la figure 6(a) est 1-bloquant. Maintenant, nous pouvons définir la classe des instances satisfaisant la propriété m-fRBTP (pour *fully Reparable Broken-Triangle Property*).

Définition 9 *Une variable x_k satisfait la propriété m-fRBTP si chaque paire de valeurs de $D(x_k)$ satisfait la propriété m-fRBTP.*

Définition 10 *Une instance binaire I dotée d'un ordre $<$ sur les variables satisfait la propriété m-fRBTP ($m \leq n-3$) par rapport à cet ordre $<$ si pour chaque variable x_k,*

- *x_k satisfait la propriété m-RBTP dans la sous-instance de I restreinte aux variables x_i telles que $x_i \leq x_k$, et*
- *x_k n'est pas m-bloquante par rapport à n'importe quelle variable $x_p > x_k$.*

L'instance de la figure 6(a) ne satisfait pas la propriété 1-fRBTP pour l'ordre sur les variables $x_\ell < x_i < x_j < x_k < x_p$ car x_k est une variable 1-bloquante. Le corollaire suivant établit la relation liant m-RBTP et m-fRBTP.

Corollaire 1 *0-fRBTP = BTP et m-fRBTP $\subsetneq$ m-RBTP $\subsetneq$ m-pRBTP.*

Le théorème 1 est une conséquence directe de la proposition 2, des lemmes 1 et 2 et du théorème 3.1 de [5].

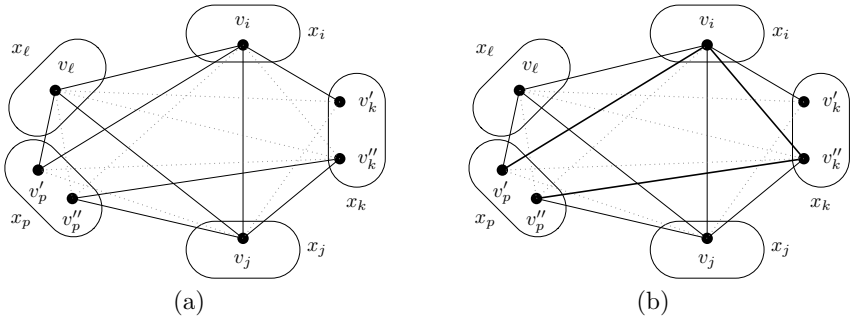


FIGURE 6 – (a) Une instance binaire I satisfaisant la propriété 1-RBTP selon l'ordre sur les variables $x_\ell < x_i < x_j < x_k < x_p$ malgré les deux triangles cassés sur x_p supportés par x_k. (b) Suite à l'ajout du tuple (v_i, v''_k) à R_{ik}, I ne respecte plus 1-RBTP (à cause du nouveau triangle cassé $(v_i, v''_k, v'_p, v''_p)$ non supporté par x_k).

Théorème 1 *Si une instance binaire $I = (X, D, R)$ satisfait la propriété m-fRBTP par rapport à un ordre $<$ donné sur ses variables, alors I peut être résolue en temps polynomial.*

Preuve : Soit une instance binaire I satisfaisant la propriété m-fRBTP par rapport à un ordre $<$. D'après la proposition 2 et les lemmes 1 et 2, nous savons que réparer un triangle cassé sur x_k ne remet pas en cause l'équivalence, ni la propriété m-fRBTP. Cette opération s'effectue en $O(n^3d^4)$. Pour cela, nous commençons par la quatrième variable selon l'ordre $<$, nous devons identifier puis réparer tous les triangles cassés sur chaque paire de valeurs. Pour chaque variable x_k, nous avons besoin de parcourir toutes les paires de valeurs, ce qui peut être fait en $O(d^2)$, et de trouver tous les triangles cassés relatifs à chacune d'elles, ce qui peut être accompli en $O(n^2d^2)$ pour une paire donnée. Au final, cette opération doit être appliquée sur les $(n-3)$ dernières variables selon l'ordre $<$. L'instance ainsi obtenue satisfait la propriété BTP par rapport à l'ordre $<$ et, donc, à ce titre, peut être résolue en $O(ed^2)$ [5]. □

Théorème 2 *Si une instance binaire I satisfait la propriété m-fRBTP par rapport à un ordre $<$ sur ses variables, alors éliminer les triangles cassés sur chacune des variables x_k (avec $k > 3$) en autorisant certains tuples interdits comme indiqué dans la section 4 est une opération de BTPisation.*

5 Interaction avec certaines classes polynomiales existantes basées sur BTP

Dans cette section, nous étudions les relations existantes entre m-fRBTP et certaines classes polynomiales connues dont la définition est inspirée de la propriété BTP.

5.1 BTP^{AC}

Nous commençons avec les classes polynomiales cachées introduites dans [8], et, en particulier, la classe BTP^{AC}. BTP^{AC} est la plus petite classe polynomiale cachée reposant sur la propriété BTP et obtenue par application d'un filtrage relatif à un niveau de cohérence donné.

Définition 11 (BTP^{AC} [8]) *Une instance binaire I satisfait BTP^{AC} si l'instance obtenue à partir de I en appliquant la cohérence d'arc*[2] *satisfait la propriété BTP.*

Nous pouvons maintenant établir le lien entre BTP^{AC} et m-pRBTP.

Théorème 3 *m-pRBTP et BTP^{AC} sont incomparables.*

Preuve : La figure 7(a) présente une instance binaire satisfaisant BTP^{AC} puisqu'après l'application de la cohérence d'arc, les triangles cassés (en rouge) ont disparu. Cette instance ne satisfait pas la propriété 1-pRBTP (ni trivialement la propriété m-pRBTP, pour $m > 1$) car x_i, x_j et x_ℓ ne peuvent être dans $\neg BT(x_k)$.

La figure 7(b) représente la microstructure d'une instance ne satisfaisant pas BTP^{AC}. En effet, chaque valeur de cette instance est arc-cohérente et on peut observer la présence d'un triangle cassé $(v'_i, v'_j, v'_k, v''_k)$. En revanche, cette instance satisfait 1-pRBTP puisque la variable x_ℓ est dans $\neg BT(x_k)$ et supporte le triangle cassé $(v'_i, v'_j, v'_k, v''_k)$. □

2. Soit une instance binaire $I = (X, D, R)$. Une valeur $v_i \in D(x_i)$ est arc-cohérente vis-à-vis de $R_{ij} \in R$ s'il existe une valeur $v_j \in D(x_j)$ telle que $(v_i, v_j) \in R_{ij}$. Un domaine $D(x_i)$ est arc-cohérent vis-à-vis de R_{ij} si, pour chaque valeur $v_i \in D(x_i)$, la valeur v_i est arc-cohérente vis-à-vis de R_{ij}. I est arc-cohérente si chaque domaine $D(x_i) \in D$, est arc-cohérent vis-à-vis de chaque relation $R_{ij} \in R$. [14]

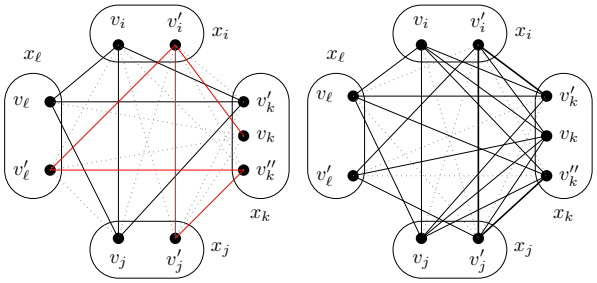


FIGURE 7 – (a) Une instance binaire satisfaisant BTP^{AC} mais qui n'est pas 1-pRBTP en considérant l'ordre $x_\ell < x_i < x_j < x_k$. (b) Une instance binaire satisfaisant la propriété 1-pRBTP, mais ne respectant pas BTP^{AC} en considérant l'ordre $x_\ell < x_i < x_j < x_k$.

5.2 k-BTP

ETP [12], et plus généralement k-BTP [6], définissent des extensions de BTP qui autorisent la présence de certains triangles cassés. Formellement, k-BTP est définie ainsi :

Définition 12 (k-BTP [6]) *Une instance binaire I satisfait la propriété k-BTP pour un entier k donné ($2 \leq k < n$) par rapport à un ordre $<$ sur ses variables si, pour tout sous-ensemble de variables $\{x_{i_1}, x_{i_2}, \ldots, x_{i_{k+1}}\}$ tel que $x_{i_1} < x_{i_2} < \ldots < x_{i_{k+1}}$, il existe au moins deux variables $(x_{i_j}, x_{i_{j'}})$ avec $1 \leq j < j' \leq k$ telles qu'il n'existe aucun triangle cassé sur x_{k+1} vis-à-vis de x_{i_j} et de $x_{i_{j'}}$.*

Définie ainsi, k-BTP ne constitue pas une classe polynomiale. En effet, la définition de la classe polynomiale correspondante nécessite par ailleurs que l'instance satisfasse la k-cohérence forte [3].

Théorème 4 *[6] Soit I une instance binaire telle qu'il existe une constante k avec $2 \leq k < n$ pour laquelle I satisfait, à la fois, la k-cohérence forte et la propriété k-BTP vis-à-vis d'un ordre $<$ sur les variables. I peut être résolue en temps polynomial.*

Comme m-fRBTP ne requiert pas de satisfaire un quelconque niveau de cohérence pour définir une classe polynomiale, nous établissons d'abord le lien entre m-pRBTP et k-BTP.

Théorème 5 *Étant données une instance binaire I dotée d'un ordre $<$ sur les variables et une constante k avec $2 \leq k < n$, si I satisfait la propriété $(k-2)$-pRBTP vis-à-vis de $<$, alors I satisfait la propriété k-BTP vis-à-vis de $<$.*

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

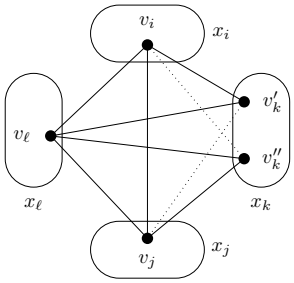


FIGURE 8 – Une instance CSP qui satisfait 3-BTP mais pas 1-pRBTP en considérant l'ordre $x_\ell < x_i < x_j < x_k$.

Preuve : Pour $k = 2$, il est clair que 0-pRBTP = 2-BTP puisque cela correspond à la propriété BTP. Pour $k > 2$, considérons une instance binaire $I = (X, D, R)$ satisfaisant $(k-2)$-pRBTP vis-à-vis de $<$ (avec $2 < k < n$). Par conséquent, pour chaque variable $x \in X$, il existe un ensemble de $r \leq k-2$ variables supports $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \subseteq \neg BT(x_k)$. Il est évident qu'aucune variable x_s avec $s < k$ et $x_s \notin \neg BT(x_k)$ ne peut contribuer à un triangle cassé avec x_ℓ ($x_\ell \in \{x_{\ell_1}, \ldots, x_{\ell_r}\}$) sur x_k. Donc, I satisfait aussi la propriété k-BTP vis-à-vis de $<$. □

La réciproque de ce théorème est fausse comme le montre la figure 8. En effet, $v'_k, v''_k \in D(x_k)$ ne satisfait pas la propriété 1-pRBTP puisqu'il existe un triangle cassé (v_i, v_j, v'_k, v''_k) et que la seule variable de $X \setminus \{x_i, x_j, x_k\}$, à savoir x_ℓ, ne peut le soutenir.

Corollaire 2 *Étant donnée une constante k avec $2 \leq k < n$, si une instance binaire fortement k-cohérente satisfait la propriété $(k-2)$-pRBTP vis-à-vis de $<$, alors elle satisfait également, la propriété k-BTP vis-à-vis de $<$.*

Le théorème 5 et le corollaire 2 sont importants car, s'il existe un ordre $<$ sur les variables pour lequel une instance binaire I satisfait la propriété k-BTP, mais n'est pas fortement k-cohérente, alors nous ne pouvons pas appliquer la k-cohérence forte pour deux raisons. D'abord, la k-cohérence est susceptible de modifier l'arité et/ou la structure de l'instance. Ensuite, la cohérence de chemin (ainsi que les cohérences plus fortes) peuvent invalider la propriété BTP (et aussi k-BTP et m-fRBTP). Ainsi, pour une instance qui satisfait à la fois k-BTP et m-fRBTP, il est possible de la résoudre en utilisant simplement la cohérence d'arc et sans avoir besoin de s'assurer que l'instance satisfait bien la k-cohérence forte.

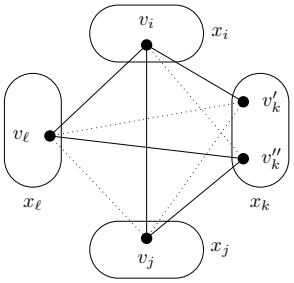


FIGURE 9 – Une variable qui satisfait la propriété 1-fBTP mais pas la propriété 1-fRBTP.

5.3 Les triangles cassés flexibles

Nous terminons cette section en établissant le lien existant avec la plus récente des classes polynomiales basées sur BTP, à savoir m-fBTP (pour Flexible Broken-Triangle Property [7]).

Définition 13 *Une paire de valeurs $v'_k, v''_k \in D(x_k)$ satisfait la propriété m-**fBTP** avec $m \leq n-3$ si, pour chaque triangle cassé (v_i, v_j, v'_k, v''_k) avec $v_i \in D(x_i)$ et $v_j \in D(x_j)$, il existe un ensemble de $r \leq m$ variables supports $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \subseteq X \setminus \{x_i, x_j, x_k\}$ tel que, pour toute affectation cohérente $(v_{\ell_1}, \ldots, v_{\ell_r}) \in D(x_{\ell_1}) \times \ldots \times D(x_{\ell_r})$, il existe $\alpha \in \{1, \ldots, r\}$ tel que si $(v_{\ell_\alpha}, v_i) \in R_{\ell_\alpha i}$, alors $(v_{\ell_\alpha}, v_j) \notin R_{\ell_\alpha j}$.*

Théorème 6 *Pour $m \geq 1$, m-fRBTP et m-fBTP sont incomparables.*

Preuve : Le couple de valeurs $v'_k, v''_k \in D(x_k)$ dans la microstructure de la figure 9 satisfait la propriété 1-fBTP, mais pas la propriété 1-fRBTP. Le couple de valeurs $v'_k, v''_k \in D(x_k)$ dans la microstructure de la figure 4(a) satisfait la propriété 1-fRBTP, mais pas la propriété 1-fBTP car on a la fois $(v_i, v_\ell) \in R_{i\ell}$ et $(v_j, v_\ell) \in R_{j\ell}$. □

6 Fusion de valeurs et élimination de variables

BTP possède plusieurs propriétés intéressantes au-delà de la polynomialité de la résolution. C'est le cas notamment de la possibilité d'éliminer des variables ou de fusionner des valeurs grâce à BTP. En effet, il a été démontré que l'absence de triangles cassés sur une variable x_k d'une instance binaire arc-cohérente I permet d'éliminer cette variable sans affecter la satisfiabilité de I [5]. L'absence de triangles cassés sur un couple de valeurs $v'_k, v''_k \in D(x_k)$ permet également de fusionner ces valeurs tout en préservant la satisfiabilité [2, 4]. Ici, nous montrons que m-RBTP permet aussi la fusion de valeurs mais pas l'élimination de variables. Ceci s'appuie sur la proposition 3. Nous rappelons, au préalable, la propriété m-wBTP [3] :

Définition 14 *Un couple de valeurs $v'_k, v''_k \in D(x_k)$ satisfait m-wBTP si pour chaque triangle cassé (v_i, v_j, v'_k, v''_k) avec $v_i \in D(x_i)$ et $v_j \in D(x_j)$, il existe un ensemble de $r \leq m$ variables $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \subseteq X \setminus \{x_i, x_j, x_k\}$ tel que pour tout $(v_{\ell_1}, \ldots, v_{\ell_r}) \in D(x_{\ell_1}) \times \ldots \times D(x_{\ell_r})$, si $(v_{\ell_1}, \ldots, v_{\ell_r}, v_i, v_j)$ est une affectation cohérente, alors il existe $\alpha \in \{1, \ldots, r\}$ tel que $(v_{\ell_\alpha}, v'_k), (v_{\ell_\alpha}, v''_k) \notin R_{\ell_\alpha k}$.*

De manière similaire, nous pouvons définir la propriété m-pRBTP pour un couple de valeurs ainsi :

Définition 15 *Un couple de valeurs $v'_k, v''_k \in D(x_k)$ satisfait m-pRBTP s'il existe un ensemble de $r \leq m$ variables $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \in \mathcal{S}_k$, tel que pour chaque triangle cassé (v_i, v_j, v'_k, v''_k) avec $v_i \in D(x_i)$ et $v_j \in D(x_j)$, pour toute affectation cohérente $(v_{\ell_1}, \ldots, v_{\ell_r}, v_i)$, avec $\forall t \in \{1, \ldots, r\}, v_{\ell_t} \in D(x_{\ell_t})$, il existe $\alpha \in \{1, \ldots, r\}$ tel que $(v_{\ell_\alpha}, v'_k) \notin R_{\ell_\alpha k}$ ou $(v_{\ell_\alpha}, v''_k) \notin R_{\ell_\alpha k}$.*

Proposition 3 *Étant données une instance binaire $I = (X, D, R)$ et une constante m (avec $0 \leq m \leq n-4$), si un couple de valeurs $v'_k, v''_k \in D(x_k)$ satisfait m-pRBTP, alors il satisfait aussi m-wBTP.*

Preuve : Par définition, il existe un ensemble $S = \{x_{\ell_1}, \ldots, x_{\ell_r}\} \in \mathcal{S}_k$ ($r \leq m$) tel que pour chaque triangle cassé (v_i, v_j, v'_k, v''_k) avec $v_i \in D(x_i)$ et $v_j \in D(x_j)$, pour toute affectation cohérente $(v_{\ell_1}, \ldots, v_{\ell_r}, v_i)$ sur $S \cup \{x_i\}$, il existe $\alpha \in \{1, \ldots, r\}$ tel que $(v_{\ell_\alpha}, v'_k) \notin R_{\ell_\alpha k}$ ou $(v_{\ell_\alpha}, v''_k) \notin R_{\ell_\alpha k}$. Donc pour chaque $v_{\ell_\alpha} \in D(x_{\ell_\alpha})$, on a :

1. soit v_{ℓ_α} est compatible avec v_j : si v_{ℓ_α} est compatible avec v'_k mais pas avec v''_k (ou inversement), x_{ℓ_α} contribue à un triangle cassé sur v'_k, v''_k, ce qui est impossible, car $x_{l_\alpha} \in S \subseteq \neg BT(x_k)$. Donc, v_{ℓ_α} ne peut être compatible avec v'_k, ni avec v''_k. Dans ce cas, $\{x_{\ell_1}, \ldots, x_{\ell_r}\}$ est un ensemble de variables supports pour le triangle cassé (v_i, v_j, v'_k, v''_k), par définition de m-wBTP.
2. soit v_{ℓ_α} n'est pas compatible avec v_j : rien n'empêche $\{x_{\ell_1}, \ldots, x_{\ell_r}\}$ d'être les variables supports de v'_k, v''_k pour m-wBTP.

Par conséquent, le couple de valeurs $v'_k, v''_k \in D(x_k)$ satisfait aussi m-wBTP. □

La réciproque de cette proposition est fausse comme le montre l'exemple de la figure 9. En effet, le couple de valeurs v'_k, v''_k satisfait la propriété 1-fBTP, et donc 1-wBTP, mais pas la propriété 1-fRBTP.

D'après la proposition 2 de [3], fusionner deux valeurs $v'_k, v''_k \in D(x_k)$ qui satisfont m-wBTP ne change pas la satisfiabilité de l'instance. En s'appuyant sur ce résultat et sur la proposition 3, nous pouvons en déduire le corollaire suivant :

Corollaire 3 *Dans une instance binaire I, fusionner un couple de valeurs $v'_k, v''_k \in D(x_k)$ qui satisfait m-pRBTP ne change pas la satisfiabilité de I.*

Comme m-pRBTP généralise BTP, et la fusion par BTP [4, 2] généralise la substitution de voisinage [10] et l'interchangeabilité virtuelle [13], nous pouvons déduire le résultat suivant :

Corollaire 4 *La fusion par m-pRBTP généralise la fusion par BTP [4, 2], la substitution de voisinage [10] et l'interchangeabilité virtuelle [13].*

Le corollaire suivant se déduit des résultats précédents.

Corollaire 5 *m-pRBTP permet de fusionner plus de valeurs que BTP et moins que m-wBTP.*

Contrairement à BTP et m-fBTP, m-pRBTP et plus particulièrement m-fRBTP n'autorisent pas l'élimination de variable car m-fBTP et m-fRBTP sont incomparables, d'après le théorème 6 et le fait que m-fBTP est une condition d'élimination de variable maximale (voir section 4 de [7]).

7 Conclusion

Dans ce papier, nous avons introduit la propriété des triangles cassés pleinement réparables, qui étend la propriété BTP en autorisant la présence de certains triangles cassés. Nous avons démontré que les instances binaires satisfaisant la propriété m-fRBTP peuvent être résolues en temps polynomial en autorisant certains tuples interdits tout en préservant l'équivalence puis en appliquant la cohérence d'arc. Ensuite, nous avons comparé cette nouvelle classe polynomiale avec certaines des classes polynomiales existantes inspirées par la propriété BTP comme BTP^{AC}, k-BTP ou m-fBTP. Finalement, nous avons prouvé que la fusion de tout couple de valeurs d'une instance binaire qui satisfait m-fRBTP ne modifie pas la satisfiabilité de l'instance. Malheureusement, m-fRBTP, comme k-BTP et m-wBTP, ne permet pas l'élimination de variables.

Différentes perspectives sont envisageables pour la poursuite de ce travail. Par exemple, d'un point de vue théorique, on pourrait s'intéresser à déterminer s'il est possible d'autoriser encore plus de tuples interdit tout en préservant l'équivalence, alors que, sur le plan pratique, on pourrait établir l'existence d'instances satisfaisant cette nouvelle propriété.

Références

[1] M. C. Cooper. Beyond consistency and substitutability. In *CP*, pages 256–271, 2014.

[2] M. C. Cooper, A. Duchein, A. El Mouelhi, G. Escamocher, C. Terrioux, and B. Zanuttini. Broken triangles : From value merging to a tractable class of general-arity constraint satisfaction problems. *Artificial Intelligence*, 234 :196 – 218, 2016.

[3] M. C. Cooper, A. El Mouelhi, and C. Terrioux. Extending broken triangles and enhanced value-merging. In *CP*, pages 173–188, 2016.

[4] M. C. Cooper, A. El Mouelhi, C. Terrioux, and B. Zanuttini. On Broken Triangles. In *CP*, pages 9–24, 2014.

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

[6] M. C. Cooper, P. Jégou, and C. Terrioux. A microstructure-based family of tractable classes for CSPs. In *CP*, pages 74–88, 2015.

[7] A. El Mouelhi. A BTP-based family of variable elimination rules for binary CSPs. In *AAAI*, pages 3871–3877, 2017.

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

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

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

[11] P. Jégou. Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In *AAAI*, pages 731–736, 1993.

[12] P. Jégou and C. Terrioux. The extendable-triple property : A new CSP tractable class beyond BTP. In *AAAI*, pages 3746–3754, 2015.

[13] C. Likitvivatanavong and R. H. C. Yap. Many-to-many interchangeable sets of values in csps. In *SAC*, pages 86–91, 2013.

[14] A. K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8 :99–118, 1977.

[15] W. Naanaa. Unifying and extending hybrid tractable classes of CSPs. *J. Exp. Theor. Artif. Intell.*, 25(4) :407–424, 2013.

[16] W. Naanaa. Extending the broken triangle property tractable class of binary CSPs. In *SETN*, pages 3 :1–3 :6, 2016.

Améliorer les méthodes de décomposition pour le dénombrement exact de solutions

Philippe Jégou **Hanan Kanso** **Cyril Terrioux**

Aix Marseille Univ, Université de Toulon, CNRS, ENSAM, LSIS, Marseille, France
{philippe.jegou, hanan.kanso, cyril.terrioux}@lsis.org

Résumé

Le problème du dénombrement de solutions d'une instance CSP, appelé #CSP, constitue un problème extrêmement difficile qui possède de multiples applications en Intelligence Artificielle. S'il est le plus souvent résolu par des méthodes approchées, ici, nous nous focalisons sur le dénombrement exact. Nous montrons comment il est possible d'améliorer les méthodes basées sur les décompositions structurelles en améliorant la recherche d'une nouvelle solution, qui est une étape essentielle, en particulier pour de telles méthodes. De plus, si les ressources en temps ou en espace sont insuffisantes, nous montrons comment notre approche est capable de fournir une borne inférieure du nombre de solutions. Des expérimentations sur des benchmarks CSP mettent en avant l'intérêt pratique de notre approche par rapport aux meilleures méthodes de la littérature.

Ce papier est un résumé de [6].

Abstract

The problem of counting solutions in CSP, called #CSP, is an extremely difficult problem that has many applications in Artificial Intelligence. This problem can be addressed by exact methods, but more classically it is solved by approximate methods. Here, we focus primarily on the exact counting. We show how it is possible to improve the methods based on structural decomposition by offering to enhance the search for a new solution which is a critical step for counting, particularly for such methods. Moreover, if the resources in time or in space are insufficient, we show that our approach is still able to provide a lower bound of the result. Experiments on CSP benchmarks show the practical advantage of our approach w.r.t. the best methods of the literature.

This is a summary of [6].

1 Contexte

Dénombrer les solutions d'une instance CSP (problème #CSP) ou les modèles d'une instance SAT (problème #SAT) constituent des problèmes extrêmement difficiles qui possèdent de multiples applications. Ces problèmes étant connus pour être #P-complet [10], leur résolution d'un point de vue pratique s'effectue souvent par le biais de méthodes approchées offrant généralement une borne inférieure du nombre de solutions. Cependant, en exploitant certaines propriétés de l'instance, il est envisageable de proposer des méthodes exactes qui soient efficaces en théorie comme en pratique [2, 3, 5, 4]. C'est à ce type d'approche que nous nous intéressons ici et plus particulièrement à la méthode #BTD [4] qui dénombre les solutions d'une instance CSP à l'aide d'une décomposition arborescente. Bien qu'ayant fait preuve de résultats pratiques intéressants, cette méthode peut être améliorée, comme nous le montrons dans la section suivante.

2 Améliorer #BTD

Une instance de CSP (pour Problème de Satisfaction de Contraintes) est définie par la donnée d'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 taille au plus d, et $C = \{c_1, \ldots, c_e\}$ est un ensemble de e contraintes. Chaque contrainte c_i est un couple $(S(c_i), R(c_i))$, où $S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ définit la *portée* de c_i, et $R(c_i) \subseteq d_{x_{i_1}} \times \cdots \times d_{x_{i_k}}$ est une *relation de compatibilité*. La structure d'une instance CSP est donnée par un hypergraphe, appelé *hypergraphe de contraintes*, dont les sommets correspondent aux variables et les arêtes aux portées des contraintes. Une affectation d'un sous-ensemble de X est dite *cohérente* si toutes les contraintes portant sur ce sous-ensemble sont satisfaites. Une *solution* est une affectation cohérente de toutes les variables.

Certaines méthodes, comme #BTD, exploitent la

notion de *décomposition arborescente de graphes* [7] pour identifier des sous-problèmes indépendants.

Définition 1 *Une* décomposition arborescente *d'un graphe $G = (X, C)$ est un couple (E, T) où $T = (I, F)$ est un arbre (I est un ensemble de nœuds et F un ensemble d'arêtes) et $E = \{E_i : i \in I\}$ une famille de sous-ensembles de X, telle que chaque sous-ensemble (appelé 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 à j dans T, alors $E_i \cap E_j \subseteq E_k$. La largeur d'une décomposition est égale à $max_{i \in I}|E_i|-1$. La largeur arborescente dite* tree-width *w de G est la largeur minimale pour toutes les décompositions arborescentes de G.*

Dans #BTD, chaque sous-problème est défini vis-à-vis d'un cluster E_i (ce sous-problème porte sur toutes les variables présentes dans la descendance de E_i) et de l'affectation courante sur les variables de l'intersection de E_i avec son père. Pour chaque sous-problème, #BTD va procéder à une exploration systématique et dénombrer le nombre de solutions locales qu'il possède avant de les combiner pour établir le nombre de solutions du problème global. Ce faisant, #BTD obtient une complexité en temps en $O(n.e.d^{w^+ +1})$ (avec w^+ la largeur de la décomposition utilisée), qui d'un point de vue théorique, est une des meilleures possibles. D'un point de vue pratique, bien qu'ayant obtenu des résultats intéressants, cette méthode peut être améliorée. En effet, pour chaque sous-problème, #BTD s'attache à dénombrer toutes ses solutions. Pour autant, à ce stade-là, il ne possède aucune garantie que les solutions locales ainsi trouvées pourront s'étendre sur le reste du problème pour former des solutions du problème global. Si ces solutions locales ne participent pas à des solutions globales, #BTD aura effectué un calcul inutile des plus coûteux compte tenu de la complexité théorique du problème #CSP. Aussi, ici, afin d'éviter cela, nous proposons de modifier #BTD de sorte que, lorsque l'algorithme trouve une solution locale pour un sous-problème donné, celui-ci s'assure d'abord que cette solution participe à une solution globale avant de dénombrer toutes les solutions de ce sous-problème. Si cette idée peut paraître simple et naturelle, elle modifie significativement le comportement de l'algorithme initial et nécessite, au final, de définir un nouvel algorithme, appelé #EBTD (pour Enhanced BTD), et un nouveau cadre formel avec notamment la notion de *goods structurels partiels.*

Théorème 1 *#EBTD a une complexité en temps en $O(n.(re+ns).d^{w^+ +1})$ pour une complexité en espace en $O(n.s.d^s)$ avec s la taille de la plus grande intersection entre deux clusters et $r = max_{c \in C}|S(c)|$.*

Par ailleurs, si les ressources en temps ou en espace sont insuffisantes, par construction, #EBTD est en mesure de fournir une borne inférieure du nombre de solutions, qui, en pratique, s'avère souvent non nulle.

Nous avons comparé expérimentalement #EBTD à Toulbar2/#BTD [4] (c'est-à-dire #BTD), Cachet [8], c2d [2], relsat [1] et sharpSAT [9] qui constituent les méthodes de référence pour la résolution des problèmes #CSP ou #SAT. Il en résulte que #EBTD résout plus d'instances (à savoir 908 instances sur les 1 059 instances considérées) que sharpSAT (899 instances), Toulbar2/#BTD (877 instances), Cachet (608 instances), relsat (618 instances) ou c2d (382 instances) tout en se révélant plus rapide sur la plupart des instances.

3 Conclusion

Nous avons proposé un nouvel algorithme, appelé #EBTD, pour calculer le nombre exact de solutions d'une instance CSP. Si la complexité de cet algorithme est similaire à celle de #BTD, en pratique, il se révèle bien plus efficace et résout plus d'instances et plus rapidement que les méthodes de l'état de l'art.

Plusieurs extensions sont envisageables comme notamment l'exploitation de décompositions adaptées au problème #CSP, ou l'utilisation de cette approche pour calculer de meilleures approximations du nombre de solutions.

Références

[1] R. Bayardo and J. Pehoushek. Counting Models Using Connected Components. In *AAAI*, pages 157–162, 2000.

[2] A. Darwiche. New Advances in Compiling CNF into Decomposable Negation Normal Form. In *ECAI*, pages 328–332, 2004.

[3] R. Dechter and R. Mateescu. The Impact of AND/OR Search Spaces on Constraint Satisfaction and Counting. In *CP*, pages 731–736, 2004.

[4] A. Favier, S. de Givry, and P. Jégou. Exploiting Problem Structure for Solution Counting. In *CP*, pages 335–343, 2009.

[5] V. Gogate and R. Dechter. Approximate Solution Sampling (and Counting) on AND/OR search spaces. In *CP*, pages 534–538, 2008.

[6] P. Jégou, H. Kanso, and C. Terrioux. Improving Exact Solution Counting for Decomposition Methods. In *ICTAI*, pages 327–334, 2016.

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

[8] T. Sang, F. Bacchus, P. Beame, H. Kautz, and T. Pitassi. Combining Component Caching and Clause Learning for Effective Model Counting. In *SAT*, 2004.

[9] M. Thurley. sharpSAT - Counting Models with Advanced Component Caching and Implicit BCP. In *SAT*, pages 424–429, 2006.

[10] L. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, 8 :189–201, 1979.

Optimisation bi-critère de la vitesse le long d'un trajet maritime

Estelle Chauveau[1,2] * **Philippe Jégou**[2] **Nicolas Prcovic**[2]

[1] Atos
73 Rue de Saint-Mandrier-sur-Mer, 83140 Six-Fours-les-Plages (France)
[2] Aix-Marseille Univ, Université de Toulon, CNRS, ENSAM, LSIS, Marseille, France
Avenue Escadrille Normandie-Niemen, 13397 Marseille Cedex 20 (France)
{estelle.chauveau, philippe.jegou, nicolas.prcovic}@lsis.org

Résumé

Avec la dépendance du prix du pétrole et ses variations ainsi que les préoccupations environnementales, les acteurs du transport maritime sont de plus en plus nombreux à chercher des outils d'aide à la décision leur permettant d'optimiser le temps de trajet en même temps que la consommation de carburant (optimisation bi-objectif). Dans le problème traité, la donnée d'entrée est un itinéraire, et nous nous intéressons à optimiser le temps de trajet et la consommation en fonction d'un unique paramètre : la vitesse le long de cet itinéraire. Nous présentons ainsi une modélisation du problème, puis nous expliquons pourquoi une approche exacte ne semble pas pertinente. Nous proposons ensuite trois méthodes de résolution qui semblent adaptées au contexte et qui devront dans un second temps être implémentés puis testés afin de juger de leur intérêt pratique.

1 Introduction

Dans le domaine du transport de marchandises par voies maritimes, les grandes compagnies d'affrètement cherchent à optimiser la consommation de carburant en même temps que le temps de transport des marchandises. Ces deux objectifs sont contradictoires puisque l'augmentation de la vitesse du bateau diminue le temps de transport mais augmente la consommation de carburant (cf figure 1). Les conditions météorologiques (notamment prévisionnelles) sont un élément à prendre en compte dans cette optimisation puisqu'il influe sur les deux objectifs. Ainsi, choisir la route qui sera empruntée par le bateau est un processus complexe. Pour cela les compagnies d'affrètement maritime recherchent des systèmes d'aide à la décision proposant rapidement un éventail d'alternatives intéressantes. Une route maritime est définie par un itinéraire ainsi que la vitesse le long de cet itinéraire. Nous nous sommes intéressés dans un premier travail [3] au calcul d'itinéraires, en fixant la vitesse à une valeur arbitraire constante. Cependant, la météo variant au cours du temps dans chaque zone, il devrait être profitable d'accélérer ou de ralentir à certains moments pour traverser les différentes zones dans des conditions globalement plus favorables que si on se déplaçait à vitesse constante. Cet article traite du problème d'optimisation des deux objectifs cités (temps de trajet et consommation de carburant) en fonction de la vitesse de navigation, et non pas de l'itinéraire. Pour cela, l'itinéraire est fixé au départ, et la vitesse est le seul paramètre du problème. Au delà de cet article, l'objectif applicatif est de proposer un outil qui combine l'optimisation de l'itinéraire avec l'optimisation de la vitesse.

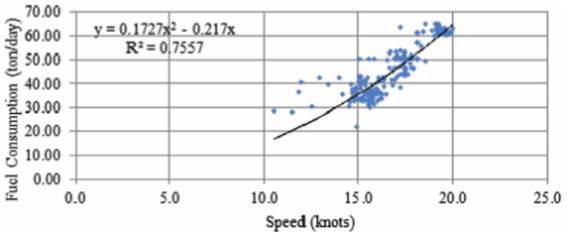


FIGURE 1 – *Allure des courbes de durée et consommation en fonction de la vitesse, à météo constante ([2]).*

*Papier doctorant : Estelle Chauveau[1,2] est auteur principal.

2 Modélisation des données du problème

Il s'agit ici de représenter des itinéraires dans le temps. À cette fin, nous allons nous appuyer sur un *graphe daté*. Dans un tel graphe, chaque sommet du graphe représente une coordonnée géographique datée (de la même manière que dans [1]) et chaque arc représente une route orthodromique entre ces coordonnées géographiques datées (date de départ et date d'arrivée). Dans ce cadre, les sommets du graphe sont les points d'un espace tridimensionnel discrétisé $E = X \times Y \times T$ où X représente la latitude avec $X = [0; x_{max}]_{\mathbb{N}}$, Y représente la longitude avec $Y = [0; y_{max}]_{\mathbb{N}}$, et T représente le temps avec $T = [0; t_{max}]_{\mathbb{N}}$. Ainsi, nous pouvons définir un graphe (S, A) où $S \subset E$ est l'ensemble des sommets et A l'ensemble des arcs. Les arcs possibles sont de la forme (s, s') tels que $s = (x, y, t)$ et $s' = (x', y', t')$ et ils doivent nécessairement vérifier deux conditions :

1. $x' = x \pm 1$ ou $y' = y \pm 1$ pour imposer que les sommets soient voisins au niveau géographique ;
2. et $t' \in [\frac{d}{v_{min}} + t; \frac{d}{v_{max}} + t]_{\mathbb{N}}$ où d est une distance telle que $d = \sqrt{(x'-x)^2 + (y'-y)^2}$ et v_{min} (resp. v_{max}) la vitesse minimale autorisée pour le bateau (resp. vitesse maximale), pour imposer que les sommets soient voisins au niveau temporel, dans le respect des vitesses autorisés.

Au graphe (S, A), nous allons adjoindre une fonction $\vec{c}$ qui associe à chaque arc un vecteur de coûts de dimension 2 ($A \to \mathbb{N}^2$). Ainsi, pour chaque arc $a_{ij} = (i, j) \in A$, on a :

$$\vec{c}(a_{ij}) = \begin{pmatrix} c_1(a_{ij}) \\ c_2(a_{ij}) \end{pmatrix}$$

où c_1 est la durée de traversée de l'arc (i, j) tandis que c_2 est la consommation de carburant sur cette traversée. Le graphe dans lequel seront exprimées les données est donc de la forme $G = (S, A, \vec{c})$.

Dans le cadre de ce travail, nous nous intéressons à l'optimisation de la vitesse le long d'un itinéraire donné. Ainsi, on appelle *itinéraire* la définition des différents points de passage dans l'espace, et *itinéraire daté* la définition des différents points de passages associés chacun à une date de passage. Les itinéraires datés correspondront donc à des chemins dans le graphe. On peut voir en fig. 2 le tracé bleu qui représente un itinéraire (le sens des arcs n'est pas indiqué ici car il est implicite). Il correspond à une projection de chemins de G dans le plan (X, Y). Le tracé orange représente les différentes vitesses possible le long d'un tel itinéraire et il s'agit donc d'itinéraires datés. On peut observer sur ce tracé orange que tous les arcs sont nécessairement «montants» car leur traversée impose un temps bien sûr strictement positif. Une conséquence immédiate est que le graphe (S, A) est nécessairement sans circuit pour des questions de temporalité.

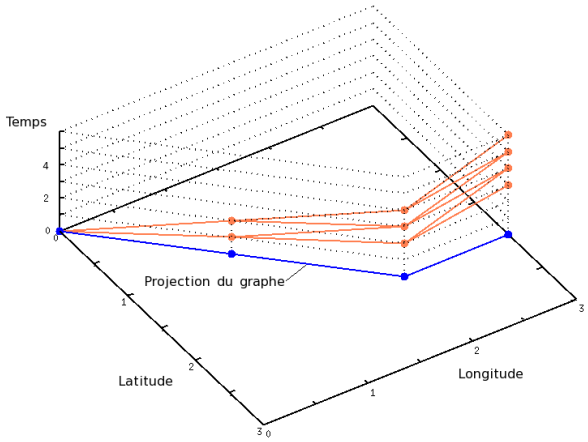


FIGURE 2 – *Représentation de chemins dans l'espace.*

Nous précisons maintenant quelques notations complémentaires utiles par la suite :

- en cas de conditions météorologiques *neutres* (pas de vent, pas de vagues etc.), nous noterons $\vec{c_1}^{neutre}(a_{ij})$ (resp. $\vec{c_2}^{neutre}(a_{ij})$) la valeur de la durée (resp. la consommation de carburant) sur l'arc a_{ij}.
- d_{ij} désigne la longueur d'un arc (i, j)
- δt la durée réelle représentée par un intervalle de temps $[t, t + 1]$
- tr_i ou *tranche géographique* d'indice i, désigne l'ensemble des arcs du graphe (courbe orange sur la fig. 2) de sommets d'origine (x_i, y_i, t) dont la projection sur le plan $X \times Y$ est le sommet (x_i, y_i) (courbe bleue sur la fig. 2) correspondant à l'unique i^{eme} arc de l'itinéraire.

Le cadre formel que nous venons de définir sur la base du graphe $G = (S, A, \vec{c})$ va nous permettre de représenter les données. Celle-ci seront en fait constituées d'un unique itinéraire mais associé à un ensemble d'itinéraires datés. Cela nous permet maintenant d'exprimer le problème que nous souhaitons traiter.

3 Problème général

Étant donné un graphe $G = (S, A, \vec{c})$ qui représente un unique itinéraire, allant du point géographique d'origine de coordonnées (x_{or}, y_{or}) jusqu'au point géographique d'arrivée (x_{ar}, y_{ar}), il s'agit de calculer des

solutions pareto-optimales par rapport aux deux objectifs considérés, soit la durée du trajet et la consommation de carburant.

3.1 Estimation de la taille du graphe daté.

Nous précisons maintenant certaines caractéristiques du graphe issues de données réelles en nous appuyant sur les expérimentations conduites dans [4]. Nous avons relevé des valeurs moyennes qui montrent que dans ce cadre, les chemins pareto-optimaux possèdent en moyenne 12 arcs tandis que la longueur moyenne d'un arc est d'environ 500 km. Par ailleurs, après consultation de certaines compagnies d'affrétement maritime, il est apparu pertinent de choisir un δt de 45 minutes et de fixer v_{min} =16 noeuds et v_{max} =20 noeuds (cela doit être ajusté pour chaque bateau en fonction de ses caractéristiques techniques). À partir de ces informations, nous avons calculé qu'il existe 5 arcs sortants à chaque noeud (cf. contraintes vues en section 2). Sur cette base, il existe alors un total de $\mathbf{5^{12}}$ **chemins différents**.

3.2 Solutions pareto-optimales

À chacun de ces chemins est associé un coût global par critère (durée et carburant consommé). Or, la durée de trajet et la consommation de carburant varient de manière opposée (en fonction de la vitesse, cf. fig. 1). Ainsi, ces deux paramètres sont antagonistes. Pour cette dernière raison, dans le cas d'une météo peu variable, le nombre de solutions pareto-optimales est du même ordre de grandeur que le nombre de trajets potentiels. Si la météo est très variable, le nombre de solutions pareto-optimales diminue mais risque de rester trop élevé pour la personne qui devra ensuite choisir la solution à suivre en pratique. En plus de cela, si deux solutions sont pareto-optimales mais sont très proches, il n'est pas nécessaire de les identifier toute les deux, une seule suffit. Aussi, plutôt que de calculer l'ensemble des chemins pareto-optimaux par une approche exacte, nous proposons dans la section 4 de restreindre le problème en nous limitant à l'identification de trajets pertinents, i.e. d'itinéraires datés qui passent par des zones dans lesquelles la météo est avantageuse.

4 Restriction du problème

4.1 Idée générale

Ce qui intéresse les compagnies maritimes peut se synthétiser par :

- la proposition d'un nombre limité de solutions (pour plus de lisibilité et pour une aide à la décision plus poussée) ;
- des chemins qui tirent partie des zones où la météo est *favorable*.

L'ensemble des chemins pareto-optimaux n'est pas une information pertinente, il s'agit ainsi de trouver les chemins qui passent au maximum par les arcs les plus *favorables* du graphe de recherche. Cette notion de *favorable* s'appuie sur un «score» que l'on va attribuer à chaque arc du graphe.

4.2 «Scorage» des arcs

Un vecteur de scores $\vec{s}(a_{ij}) = (s_1(a_{ij}), s_2(a_{ij}))$ est attribué à chaque arc a_{ij}. Ce score correspond à un pourcentage du coût de l'arc par rapport à une météo neutre. Ainsi, pour un critère donné, un score proche de 0% correspondra à une météo quasi-neutre (pas de vent, pas de courant, etc.), un score négatif à une météo avantageuse et un score positif à une météo désavantageuse. Le score de l'arc a_{ij} en considérant le critère 1 (ce qui vaut pour le critère 2) est :

$$s_1(a_{ij}) = 100 \times \frac{c_1(a_{ij}) - c_1^{neutre}(a_{ij})}{d_{ij} \times c_1^{neutre}(a_{ij})} \qquad (1)$$

avec $v(a_{ij})$ la vitesse sur l'arc a_{ij}. Le dénominateur d_{ij} permet de normer le coût de l'arc par rapport à sa longueur. Prenons un exemple avec le critère consommation de carburant, pour un arc a_{ij} dont le coût est 150 (kilos de fuel par km). Si le coût neutre (pas de vent, pas de vagues, etc.) à la vitesse $v(a_{ij})$ est de 140 (kilos de fuel par km), alors le score $s_2(a_{ij})$ vaut $100 \times (150-140)/140 \approx +7\%$ (ici d_{ij} est égal à 1). Cela correspond bien à une augmentation de la consommation par rapport à une situation de mer calme. Il s'agit alors de rechercher les chemins pertinents dans le graphe $G' = (S, A, \vec{s})$ qui ne diffère du graphe originel $G = (S, A, \vec{c})$ que par sa fonction de coût où $\vec{s} : A \to \mathbb{Q}^2$. Pour résoudre ce problème, nous envisageons trois méthodes qui devront être implémentées et testées :

- **Méthode par combinaison linéaire classique.** On identifie dans G' le plus court chemin mono-objectif d'une combinaison linéaire des deux objectifs. Le sélection des coefficients de la combinaison linéaire peut être faite en utilisant une stratégie adaptative comme proposé dans [5]. Cette méthode permet de trouver les solutions sur l'enveloppe convexe du front de pareto.
- **Méthode basée sur le critère de Tchebycheff.** Cette méthode permet de trouver des solutions même lorsqu'elles n'appartiennent pas à l'enveloppe convexe du front de pareto. On pourra s'appuyer sur les travaux présentés dans [6].
- **Méthode par élitisme** Cette dernière méthode consiste à sélectionner parmi les arcs les mieux scorés un ensemble d'*élites*. Il s'agit ensuite de rechercher les chemins qui passent par un maximum de ces arcs élites. L'avantage de cette mé-

thode est qu'elle exploite la structure du graphe (graphe orienté sans circuit).

Cette méthode est détaillée dans la section 4.3. Les résultats de ces différentes approches devront être comparés (temps de calcul et qualité des solutions identifiées).

4.3 Recherche d'un chemin pertinent par élitisme

L'objectif de cette méthode est de trouver les chemins qui empruntent les zones de météo avantageuses, en se basant sur la structure du graphe (graphe orienté sans circuit). Pour atteindre cet objectif, nous procèdons en deux étapes. La première étape consiste à identifier pour chaque critère les arcs élites en se basant, d'une part, sur le coût de ces arcs, et d'autre part, sur la tranche géographique dans laquelle l'arc se trouve. La seconde étape est constituée par la recherche de chemins passant par un maximum d'arcs élites.

Étape 1 : identification des arcs «élites». Nous introduisons ici un paramètre p qui représente la proportion d'arcs élites par tranche géographique pour chaque critère. Soit e l'arrondi à l'entier supérieur de $p \times |tr_{courante}|$. Pour un critère donné, les e meilleurs arcs de cette tranche seront considérés comme élites, marqués comme *favorables* et notés -1. Les autres arcs de cette tranche pour ce même critère étant marqués comme *neutres* et notés 0. Cette approche permet de s'assurer de l'existence d'arcs favorables au niveau d'une tranche géographique, quand bien même les conditions météo seraient mauvaises sur celle-ci. Une fois cette notation bicritère établie, nous calculons une unique pondération par arc en calculant la somme des notes attribuées pour chaque critère (-1, 0). Nous obtenons ainsi une valuation unique appartenant à $\{-2, -1, 0\}$. Il existe donc les élites classiques notées -1, et les «super» élites notées -2.

Étape 2 : calcul des chemins de meilleurs compromis. La recherche d'un plus court chemin dans un graphe orienté sans circuit à pondérations positives et négatives se fait en $O(|S| + |A|)$ ([7]), et ce malgré la présence de poids négatifs. L'algorithme efficace pour ce type de graphe consiste dans un premier temps à faire un tri topologique sur le graphe en question. Cette étape est triviale dans notre cas puisqu'il suffit d'ordonner les sommets par tranches géographiques. Cela étant, si un chemin optimal est simple à obtenir, il s'agira de les lister jusqu'à ce que l'opérateur humain considère avoir obtenu une solution suffisamment satisfaisante, le passage d'une solution à une autre étant très simple à calculer.

5 Conclusion

Dans le cadre de l'optimisation des routes maritimes, nous travaillons sur le problème de l'optimisation de la durée de trajet et de la consommation de carburant en considérant un unique paramètre : la vitesse le long d'un itinéraire donné. Ce problème a peu été étudié dans la littérature. Comme la combinatoire reste élevée et que le nombre de solutions pareto-optimales est potentiellement trop grand pour être exploitable par un décideur humain, nous nous intéressons aux méthodes qui ne fournissent pas l'intégralité des solutions pareto-optimales. Ainsi, nous évoquons trois méthodes :

- la combinaison linéaire des deux critères [5] ;
- la méthode de Lucie Galand [6] basée sur le critère de Tchebycheff ;
- la méthode par élitisme.

La finalité de ce travail est d'implémenter ces trois méthodes sur des instances réelles, et d'analyser la qualité des solutions identifiées. L'outil que nous développons devra être basé sur une ou plusieurs de ces méthodes (si les solutions qu'elles fournissent sont complémentaires). Enfin, les méthodes choisies pourront servir de sous-routine pour l'optimisation en fonction des deux paramètres vitesse et itinéraire.

Références

[1] Angelica Andersson. Multi-objective optimisation of ship routes. Master thesis, Goteborg, Sweden, Chalmer University of Technology, 2015.

[2] Nicolas Bialystocki and Dimitris Konovessis. On the estimation of ship's fuel consumption and speed curve : A statistical approach. *Journal of Ocean Engineering and Science*, 1(2) :157 – 166, 2016.

[3] Estelle Chauveau, Philippe Jégou, and Nicolas Prcovic. Un algorithme multicritère pour l'optimisation des routes maritimes en temps réel. *Journée Transports Intelligents, RFIA*, 2016.

[4] Estelle Chauveau, Philippe Jégou, and Nicolas Prcovic. Multiobjective shortest path problem in a time dependent graph (en soumission). 2017.

[5] Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle. Adaptive "anytime" two-phase local search. In *International Conference on Learning and Intelligent Optimization*, pages 52–67. Springer, 2010.

[6] Lucie Galand. Recherche d'un chemin de meilleur compromis dans un graphe multicritère. In *7ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2006)*, pages 121–136, Lille, France, February 2006. Presses Universitaires de Valenciennes.

[7] Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson and Clifford Stein. *Introduction to algorithms.* MIT Press, 2009.

Vérification de chaînes de Markov à intervalles paramétrés avec des contraintes

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

[1]Institut Mines-Télécom Atlantique - LS2N, UMR 6004 - Nantes, France
[2]Université de Nantes - LS2N, UMR 6004 - Nantes, France
[1]anicet.bart@imt-atlantique.fr [2]<nom>.<prenom>@univ-nantes.fr

Résumé

Les chaînes de Markov à intervalles paramétrés (PIMCs) sont un formalisme de spécification permettant de représenter de façon compacte des ensembles infinis de chaînes de Markov (MCs). Alors que les chaînes de Markovs sont largement utilisées pour modéliser une très grand variété de systèmes basés sur des transitions probabilistes (ex : protocoles aléatoires, systèmes biologiques, environnements financiers) les PIMCs prennent en compte l'imprécision ou le manque de connaissances quant à la probabilité exacte de chaque évènement/transition du système en considérant des intervalles paramétrés de probabilités. Dans le contexte de la vérification de systèmes probabilistes, [9] introduit trois propriétés fondamentales à vérifier sur les PIMCs et propose pour chacune d'elle un modèle de contraintes linéaires pour y répondre. Dans ce papier, nous proposons de nouveaux modèles de contraintes répondant aux mêmes propriétés et montrons les gains qu'ils apportent sur les plans théoriques et pratiques par rapport à [9] : à savoir que nos modèles en contraintes sont drastiquement plus petits et que le temps ainsi que l'espace nécessaires à la résolution se voient nettement réduits en pratique.

Abstract

Parametric Interval Markov Chains (pIMCs) are a specification formalism for representing infinite sets of Markov Chains (MCs) in a finite model. While MCs are widely used for modelling systems in which probabilities play a fundamental role, such as randomized protocols or biological systems, pIMCs take into account imprecision in the probability values : the transitions of the system are (parametric) intervals of probabilities. The state of the art models for verifying the fundamental properties over pIMCs have been introduced in [9]. In this work, we introduce, implement and test new constraint programming models for verifying these fundamental properties. We show that all our models are complete and much smaller in size than state of the art models. We propose an implementation for our constraint models. Experiments show that our models drastically reduce the resolution time and space.

1 Introduction

Un processus aléatoire est un système qui peut changer d'état en fonction d'évènements aléatoires. Parmi ces processus, les chaînes de Markov (MCs) représentent des processus avec pas ou peu de mémoire : l'évolution du processus ne dépend que de l'état courant (ou d'une mémoire bornée dans le temps) et pas de tout son historique. Dans la suite, nous nous intéressons aux cas où le temps est discrétisé, les *Discrete Time Markov Chains* (DTMC or MC for short). A chaque pas, le processus passe de l'état courant à l'un de ses états successeurs selon une certaine distribution de probablités. Depuis plus d'un siècle, les MCs permettent de modéliser des systèmes complexes dans de nombreux domaines avec des applications multiples : prédictions de réactions chimiques, *ranking* de pages web, reconnaissance de la parole, prévisions économiques, composition musicale, etc (voir par exemple [10, 12, 18]).

Modéliser une application comme une chaîne de Markov suppose de connaître exactement la probabilité de chaque transition du système. Cependant, ces probabilités peuvent être difficiles à calculer ou à mesurer dans le cas d'applications réelles (*e.g.*, erreurs de mesure, connaissance imparfaite). Les chaînes de Markov à intervalles, ou Interval Markov Chains (IMCs), introduites en 1991 par Larsen and Jonsson [13], étendent les chaînes de Markov en remplaçant les

*Papier doctorant : Anicet Bart[1] est auteur principal.

La recherche décrite dans cet article a été partiellement financée par le projet ANR Coverif 15-CE25-0002-03.

probabilités sur les transitions par des intervalles. Ces intervalles permettent d'encadrer les valeurs possibles pour les probabilités de transitions, ce qui garantit la validité du modèle (*i.e.*, le comportement du système réel est un des comportements possible décrit par le modèle). Ainsi, une IMC est une *abstraction* qui modélise potentiellement une infinité de MCs *concrètes*. Récemment, [9] a introduit le modèle des chaînes de Markov à intervalles paramétrés (pIMCs), pour traiter le cas des intervalles dont les bornes ne sont pas connues. Chaque probabilité de transition appartient à un intervalle avec des bornes éventuellement paramétriques. Cela permet de représenter de façon compacte potentiellement une infinité de IMCs et donc de représenter dans un seul objet plusieurs niveaux de précision. Vérifier directement une IMC (ou une pIMC) garantit que les propriétés voulues soient vérifiées dans toutes les MCs *concrètes* associées. Dans le cas des IMCs, le problème du *model checking* a été traité par [4,5,11]. Dans [4,5], les auteurs proposent des algorithmes et des bornes de la complexité pour la vérification de propriétés ω-regular et LTL respectivement, sur des IMCs avec des intervalles clos. Dans cet article, nous nous intéressons à la vérification des trois propriétés suivantes sur les pIMCs :

— **Consistance :** Une pIMC donnée peut-elle être *concrétisée* en une MC ?

— **Atteignabilité existentielle :** Pour une pIMC donnée, y a-t-il une MC *concrète* atteignant un état donné ?

— **Atteignabilité universelle :** Pour une pIMC donnée, est-ce que toutes les MCs *concrètes* atteignent un état donné ?

Les modèles existant dans l'état de l'art pour vérifier ces trois propriétés ont été présentés dans [9] et en proposent la synthèse des valeurs de paramètres. Les auteurs combinent des contraintes linéaires (égalités et inégalités) avec des conjonctions et disjonctions. Le nombre de contraintes est exponentiel en le nombre d'états et de transitions de la pIMC à vérifier.

Dans cet article, nous présentons et évaluons de nouveaux modèles à base de contraintes pour vérifier ces trois propriétés fondamentales dans le cas des pIMCs. Sur le plan théorique, nous montrons que tous nos modèles sont complets et beaucoup plus petits que ceux de l'état de l'art (linéaires et non exponentiels). Sur le plan pratique, autant que nous le sachions, les modèles de l'état de l'art n'ont jamais été implémentés. Nous proposons la première implémentation de ces modèles, ainsi qu'une implémentation de nos modèles, ce qui nous permet de comparer les deux approches. Nous testons ces deux implémentations sur un large *benchmark* issu de la communauté des MCs. Nos résultats montrent que notre modélisation améliore significativement la résolution, en temps et en espace, comparée à celle de [9].

Programmation par Contraintes Notre modélisation utilise le paradigme de programmation déclarative dit programmation par contraintes, ou *Constraint programming* (CP), qui permet de modéliser une large gamme de problèmes en séparant leur sémantique (ou modèle) de la résolution. Un problème de contraintes est donné par une liste de variables, chacune à valeur dans un domaine donné, et une liste de contraintes sur ces variables. Le problème est dit satisfiable s'il existe une valuation des variables dans les domaines, satisfaisant toutes les contraintes. Pour un problème donné, un solveur de contraintes répond *oui* si le problème est satisfiable, et dans ce cas il exhibe une solution (ou toutes les solutions), et *non* sinon. Le format des problèmes étant combinatoire par construction, la résolution peut être lente. Dans certain cas, on peut convertir les problèmes dans des langages spécifiques pour accélerer la résolution (*e.g.*, LP, MILP, SMT, SAT).

2 Préliminaires

Dans cette section, nous présentons le contexte des chaînes de Markov à temps discret, puis nous introduisons les chaînes de Markov à intervalles et à intervalles paramétrés.

2.1 Chaînes de Markov à temps discret

Un processus aléatoire est un système dans lequel le passage d'un état à un autre état est probabiliste (*i.e.*, chaque état successeur a une certaine probabilité d'être choisi). Une chaîne de Makov à temps discret est un processus aléatoire dont le passage d'un état à un autre se fait par pas de temps. Dans ce contexte, chaque état est associé à une probabilité de transition sur ses successeurs potentiels.

Definition 1 (Chaîne de Markov à temps discret)**.** Une chaîne de Markov à temps discret (DTMC ou MC) est un n-uplet $\mathcal{M} = (S, s_0, p, A, V)$, où S est un ensemble fini d'états contenant un état initial s_0, un ensemble de propositions atomiques A, une fonction d'étiquetage $V : S \mapsto 2^A$, et une fonction de transition probabiliste $p : S \times S \mapsto [0, 1]$ telle que $\forall s \in S,\ \Sigma_{s' \in S} p(s, s') = 1$.

Une chaîne de Markov peut être vue comme un graphe dirigé dont les noeuds correspondent aux états de la MC et les arêtes sont étiquetées par les probabilités de la fonction de transition. Avec cette représentation, une transition manquante entre deux états est représentée par une probabilité de 0. Nous utiliserons

donc si besoin le vocabulaire des graphes dirigés. Par exemple, pour une MC $\mathcal{M}$, nous appelons *chemin* une suite finit d'états de $\mathcal{M}$. Comme il est d'usage, pour un ensemble S nous notons S^* l'ensemble $\{S^n \mid n \in \mathbb{N}\}$ (*i.e.*, l'ensemble des chemins sur S). Nous omettons en général l'opérateur de produit cartésien dans le contexte de chemins, *e.g.*, $AB^*C = \{(a, b_1, \ldots, b_n, c) \in A \times B^* \times C \mid n \in \mathbb{N}\}$.

Example 1. La figure 1 montre une MC $\mathcal{M}_1 = (S, s_0, p, A, V)$ où $S = A = \{0, 1, 2, 3, 4\}$, l'état initial est 0, et la fonction d'étiquetage est l'identité : $V = \{(0,0), (1,1), (2,2), (3,3), (4,4)\}$. On indique explicitement quelques unes des probabilités de transition, par exemple, $p(0,1) = 0.7$, $p(1,1) = 0.5$, et $p(2,4) = 0$. On déduit des transitions manquantes que $p(0,0) = 0$, $p(0,3) = 0$, $p(1,4) = 0$, etc.

Dans la suite, nous considèrerons souvent les prédécesseurs (Pred), et les successeurs (Succ) d'un état. Pour une MC sur un ensemble d'états S, un état $s \in S$ et un sous-ensemble S' de S, nous notons :

- $\texttt{Pred}(s) = \{s' \in S \mid p(s', s) \neq 0\}$,
- $\texttt{Succ}(s) = \{s' \in S \mid p(s, s') \neq 0\}$,
- $\texttt{Pred}(S') = \bigcup_{s' \in S'} \texttt{Pred}(s')$,
- $\texttt{Succ}(S') = \bigcup_{s' \in S'} \texttt{Succ}(s')$,

Example 2 (Suite de l'exemple 1)**.** Sur la figure 1, l'ensemble $\texttt{Pred}(\{0\})$ des prédécesseurs de l'état 0 est vide [1] ; l'ensemble $\texttt{Pred}(\{4\})$ des prédécesseurs de l'état 4 est $\{4\}$; l'ensemble $\texttt{Succ}(\{2\})$ des successeurs de l'état 2 est $\{1, 2\}$; et l'ensemble $\texttt{Succ}(\{1, 2\})$ des états 1 et 2 est $\{1, 2, 3\}$.

Pour une chaîne de Markov, la probabilité d'atteindre un état donné à partir de l'état initial est appelée *atteignabilité*. Comme expliqué ci-dessus, un *chemin* (ou *trace d'exécution*) est une succession d'états obtenue par une exécution de la MC un nombre fini de pas de temps. Avec notre définition de MC, chaque chemin fini corresond à au plus une exécution finie de la MC correspondante. La probabilité d'un chemin donné est le produit des probabilités des transitions choisies pendant l'exécution. Inuitivement, la probabilité d'*atteignabilité* est la somme des probabilités des chemins finis s'arrêtant sur cet état. Formellement, nous proposons la définition suivante.

Definition 2 (Atteignabilité)**.** Soit $\mathcal{M} = (S, s_0, p, A, V)$ une MC, s un état de S, et $w = (x_0, \ldots, x_n)$ un chemin dans S^*. La probabilité du chemin w, noté $p(w)$, est $\Pi_{i=0}^{n-1} p(x_i, x_{i+1})$ si x_0 est s_0 et 0 sinon. La probabilité d'atteindre l'état s, notée $p_{reach}(s)$, est 1 si s est l'état initial s_0 et $\Sigma_{w \in (S \setminus \{s\})^* \{s\}} p(w)$ sinon. On dit que l'état s est atteignable dans $\mathcal{M}$ ssi $p_{reach}(s) \neq 0$. Soit $G \subseteq S$ un ensemble d'états. On dit que G est atteignable dans $\mathcal{M}$ ssi il existe un état s dans G atteignable dans $\mathcal{M}$.

Example 3 (Suite de l'exemple 2)**.** Dans l'exemple figure 1, la probabilité du chemin (0, 2, 1, 1, 3) est le produit des probabilités 0.3, 0.5, 0.5, et 0.5, soit 0.0375. La probabilité d'atteindre l'état initial 0 est 1 (par définition). On a $p_{reach}(1) = p(0,1) + \Sigma_{i=0}^{+\infty} p(0,2) \times p(2,2)^i \times p(2,1) = p(0,1) + p(0,2) \times p(2,1) \times (1/(1 - p(2,2))) = 1$.

2.2 Abstractions de chaînes de Markov

Modéliser une application comme une chaîne de Markov suppose de connaître exactement les probabilités pour chaque transition du système. Cependant, ces quantités peuvent être difficile à calculer ou à mesurer pour des applications réelles (*e.g.*, erreurs de mesure, connaissance partielle). Les chaînes de Markov à intervalles (IMCs), introduites par Larsen et Jonsson en 1991 [13], étendent les chaînes de Markov en autorisant les probabilités de transition à varier dans des intervalles donnés. Dans la suite, on note $\mathsf{Int}_{[a,b]}$ l'ensemble contenant tous les intervalles fermés de $[a, b]$.

Definition 3 (Chaînes de Markov à intervalles [13])**.** Une chaîne de Markov à intervalles (IMC) est un n-uplet $\mathcal{I} = (S, s_0, P, A, V)$, avec S, s_0, A, et V définis précédemment, et $P : S \times S \mapsto \mathsf{Int}_{[0,1]}$ une contrainte de transition associant à chaque transition potentielle un intervalle de probabilités.

Comme pour les chaînes de Markov, une chaîne de Markov à intervalles peut être représentée par un graphe dirigé, dont les arêtes sont étiquetées par des intervalles de probabilités (au lieu de probabilités).

Example 4. La figure 2 illustre la chaîne $\mathcal{I} = (S, s_0, P, A, V)$ où S, s_0, A, et V sont identiques à la MC de la figure 1. En observant l'étiquetage, on voit que $P(0,1) = [0,1]$, $P(1,1) = [0.5,1]$, et $P(3,3) = [1,1]$. Les intervalles de probabilité des transitions manquantes sont réduits à $[0,0]$, ainsi $P(0,0) = [0,0]$, $P(0,3) = [0,0]$, $P(1,4) = [0,0]$.

Dans [13], les auteurs introduisent le couplage d'une IMC avec une relation de *satisfaction*. Cette relation permet de relier les MCs *concrètes* à leurs abstractions potentielles en IMCs, et réciproquement. Notons que la notion de satisfaction introduite par [13] opère comme une simulation basée sur les probabilités de transition et les étiquettes des états. Cela ne dépend donc pas de la structure des MC / IMC considérées.

1. Ici l'état initial n'a pas de prédécesseurs, ce qui n'est pas vrai dans le cas général.

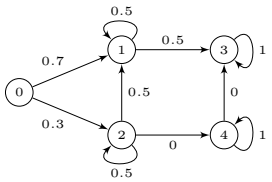


FIGURE 1 – Exemple d'une MC $\mathcal{M}_1$

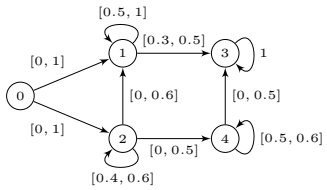


FIGURE 2 – Exemple d'une IMC $\mathcal{I}$

Definition 4 (Relation de satisfaction [13])**.** Soit $\mathcal{M} = (T, t_0, p, A, V^M)$ une MC et $\mathcal{I} = (S, s_0, P, A, V^I)$ une IMC. Une relation $\mathcal{R} \subseteq T \times S$ est une relation de satisfaction ssi pour tout $t\ \mathcal{R}\ s$ dont dont les étiquettes de s et t concordent (*i.e.*, $V^M(t) = V^I(s)$), il existe une correspondance $\delta : T \mapsto (S \mapsto [0,1])$ t.q.

1. $\forall t' \in T$ si $p(t,t') > 0$ alors $\delta(t')$ est une distribution sur S,
2. $\forall s' \in S : (\Sigma_{t' \in T} p(t,t') \cdot \delta(t')(s')) \in P(s,s')$, et
3. $\forall t' \in T, \forall s' \in S$, si $\delta(t')(s') > 0$, alors $(t', s') \in \mathcal{R}$.

On dit que l'état $t \in T$ satisfait l'état $s \in S$ (noté $t \models s$) ssi il existe une relation de satisfaction contenant (t, s). De façon similaire, on dit que $\mathcal{M}$ satisfait $\mathcal{I}$ (noté $\mathcal{M} \models \mathcal{I}$) ssi $t_0 \models s_0$.

Par définition, la relation de satisfaction entre MCs et IMCs présentée en Def. 4 autorise les implémentations et spécifications à avoir des structures non isomorphes. Pour autant, nous utiliserons dans la suite de cet article des résultats de [9] qui permettent de limiter notre étude aux MCs concrètes dont la structure est isomorphe à celle de l'IMC abstraite en toute généralité.

Example 5 (Suite de l'exemple 4)**.** Les chaînes de Markov $\mathcal{M}_1$ et $\mathcal{M}_2$ données dans les Figures 1 et 3 satisfont toutes les deux l'IMC $\mathcal{I}$ donnée dans la Figure 2. Il est clair que $\mathcal{M}_1$ a une structure isomorphe à celle de $\mathcal{I}$. Ce n'est pas le cas, en revanche, de $\mathcal{M}_2$, dans laquelle l'état 3 de $\mathcal{I}$ a été éclaté en deux états 3 et 3′.

Les chaînes de Markov à intervalles paramétrées (pIMCs), introduites dans [9], autorisent l'utilisation d'intervalles dont les bornes sont variables. Ces bornes variables sont alors représentées par des paramètres (ou des combinaisons de paramètres), ce qui permet notamment l'expression de dépendances entre plusieurs transitions du systèmes. La modification des valeurs des paramètres engendre alors des changements dans l'ensemble des chaînes de Markov qui satisfont les IMCs abstraites. En conséquence, les pIMCs permettent de représenter, d'une manière compacte et avec une structure finie, un ensemble potentiellement infini d'IMCs.

Afin de définir formellement les pIMCs, nous commençons par introduire des notations concernant les paramètres. Un paramètre $p \in Y$ est une variable dont le domaine est l'intervalle $[0,1]$. L'ensemble de tous les intervalles paramétrés fermés de la forme $[a,b]$ où a et b sont soit des constantes réelles dans $[0,1]$, soit des expressions linéaires de paramètres de Y est noté $\mathsf{Int}_{[0,1]}(Y)$. Par exemple, les intervalles $[0, 0.5]$, $[p, 1]$, $[0.3, p]$ et $[p, q]$ appartiennent tous à l'ensemble $\mathsf{Int}_{[0,1]}(\{p, q\})$.

Definition 5 (Chaîne de Markov à intervalles paramétrés)**.** Une chaîne de Markov à intervalles paramétrés est un n-uplet $\mathcal{P} = (S, s_0, P, A, V, Y)$, avec S, s_0, A et V définis comme dans les IMCs, Y un ensemble de variables (paramètres) sur le domaine $[0,1]$, et une relation de transition $P : S \times S \mapsto \mathsf{Int}_{[0,1]}(Y)$ qui associe à chaque transition potentielle un intervalle paramétré.

Etant données une pIMC $\mathcal{P} = (S, s_0, P, A, V, Y)$ et une valuation $v : Y \mapsto [0,1]$ des paramètres, l'IMC (S, s_0, P_v, A, V) obtenue en remplaçant la fonction de transition P de $\mathcal{P}$ par la fonction $P_v : S \times S \mapsto \mathsf{Int}_{[0,1]}$ définie par $\forall s, s' \in S, P_v(s,s') = v(P(s,s'))$ (*i.e.*, en remplaçant les paramètres par leur valeur donnée par v) est notée $v(\mathcal{P})$. Cette IMC est appelée une *instance* de la pIMC $\mathcal{P}$. Pour finir, une MC $\mathcal{M}$ satisfait une pIMC $\mathcal{P}$, notée $\mathcal{M} \models \mathcal{P}$, ssi il existe une IMC $\mathcal{I}$, instance de $\mathcal{P}$, telle que $\mathcal{M} \models \mathcal{I}$ au sens de la définition 4.

Example 6. Considérons la pIMC $\mathcal{P} = (S, 0, P, A, V, Y)$ donnée en Figure 4. Son ensemble d'états S contient cinq éléments 0, 1, 2, 3, et 4, avec 0 l'état initial. Pour simplifier, nous considérons que l'ensemble des labels A est égal à S et que l'étiquetage est la fonction identité. L'ensemble de paramètres Y a deux éléments p et q, et les intervalles paramétrés de la fonction de transition P sont donnés par l'étiquetage des arcs (*e.g.*, $P(1,3) = [0.3, q]$, $P(2,4) = [0, 0.5]$, et $P(3,3) = [1,1]$). Notons que l'IMC $\mathcal{I}$ donnée dans la Figure 2 est une instance de $\mathcal{P}$, dans laquelle p et q prennent respectivement les valeurs 0.6 et 0.5. De plus, comme dit dans l'exemple 5, les MC $\mathcal{M}_1$ et $\mathcal{M}_2$ (données dans les figures 1 et 3) satisfont $\mathcal{I}$. En conséquence, $\mathcal{M}_1$ et $\mathcal{M}_2$ satisfont aussi $\mathcal{P}$.

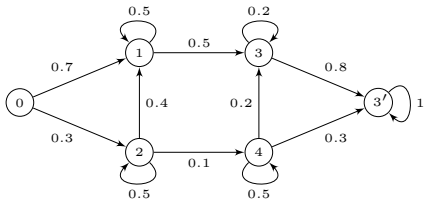


FIGURE 3 – MC $\mathcal{M}_2$ qui satisfait l'IMC $\mathcal{I}$ donnée dans Fig. 2

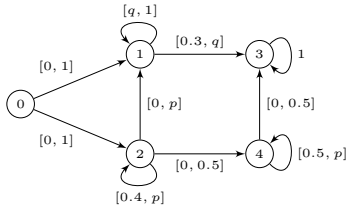


FIGURE 4 – Exemple d'une pIMC $\mathcal{P}$

Dans la suite de l'article, nous appellerons la *taille* d'une pIMC la somme de son nombre d'états et de son nombre de transitions non réduites à $[0,0]$. Pour finir, nous étendons les notations définies précédemment pour les MCs aux pIMCs. Soit une pIMC $\mathcal{P}$ dont l'ensemble d'états est S, soit un état $s \in S$ et un sous-ensemble S' de S. Nous proposons les notations suivantes :

— $\texttt{Pred}(s) = \{s' \in S \mid P(s', s) \neq [0,0]\}$,
— $\texttt{Succ}(s) = \{s' \in S \mid P(s, s') \neq [0,0]\}$,
— $\texttt{Pred}(S') = \bigcup_{s' \in S'} \texttt{Pred}(s')$,
— $\texttt{Succ}(S') = \bigcup_{s' \in S'} \texttt{Succ}(s')$,

2.3 Programmation par contraintes

La programmation par contraintes (CP) est un paradigme de programmation déclaratif dans lequel un programme consiste en une liste de variables, chacune équipée d'un domaine propre, ainsi qu'une liste de contraintes sur ces variables. Utiliser ce programme en entrée d'un solveur de contraintes produira un résultat positif si le programme est satisfiable (*i.e.*, s'il existe une valuation des variables satisfaisant les contraintes) et un résultat négatif dans le cas contraire. [2] La programmation par contrainte offre une manière élégante de modéliser les problèmes où interviennent des contraintes en mettant l'accent sur la sémantique des problèmes plutôt que sur la façon dont ces problèmes sont résolus. Lorsque le problème est satisfiable, le solveur peut générer une valuation des variables qui satisfait l'ensemble des contraintes. Cependant, étant donné qu'il est en général difficile de vérifier qu'un CP est satisfiable (*e.g.*, NP-complet dans le cas général d'inégalités entre polynomes réels), le temps de réponse des solveurs peut être très long.

Definition 6 (Problème de Satisfaction de Contraintes)**.** Un problème de satisfaction de contraintes (CSP) est un n-uplet $\Omega = (X, D, C)$ où $X = \{x_1, \ldots, x_k\}$ est un ensemble fini de variables telles que chaque $x \in X$ est associée à un domaine D_x ; D est l'ensemble des domaines associés à toutes les variables de X ; C est l'ensemble des contraintes sur les variables de X. Une valuation v pour Ω est un ensemble $v = \{(x, d) | x \in X, d \in D_x\}$ de valuations élémentaires (x, d), où, pour chaque $x \in X$ il existe une unique paire de la forme (x, d) in v. Une valuation satisfait Ω si et seulement si elle satisfait toutes les contraintes de C. Le résultat (vrai ou faux) de l'évaluation de l'ensemble des contraintes de C pour la valuation v est noté $v(C)$.

Example 7. Les deux CSPs présentés ci-après encodent le fait qu'un ensemble X de n variables $\{p_1, \ldots, p_n\}$ forme une distribution de probabilités. Soit Ω_1 le CSP $(X, \{(p, [0,1]) \mid p \in X\}, \{p_1 + \ldots + p_n = 1\}$. Ω_1 utilise n variables ayant le domaine $[0,1]$ et une unique contrainte pour s'assurer que les valeurs des variables forment une distribution de probabilité. Ce même problème pourrait aussi être encodé avec le CSP $\Omega_2 = (X, \{(p, \mathbb{R}) \mid p \in X\}, \{p_1 \geq 0, \ldots, p_n \geq 0, p_1 + \ldots + p_n = 1\}$.

Dans la suite du document, une modélisation CSP est une stratégie d'encodage d'un problème donné en CSP. Un modèle est alors un CSP obtenu par l'utilisation concrète d'une modélisation CSP. La taille d'un modèle correspond à la somme du nombre de variables et du nombre de contraintes qu'il comprend. Notons que, dans le cadre gnéral de la programmation par contraintes, le fait d'introduire moins de variables ou moins de contraintes lors du processus de modélisation n'implique pas nécéssairement une résolution plus rapide des modèles. Cependant, étant donné que les modèles proposés dans l'état de l'art dans le cadre des problèmes que nous considérons sont exponentiels dans la taille des pIMCs originales et que nous proposons ici de nouveaux modèles de taille linéaire, un gain conséquent en termes de complexité temporelle et spatiale lors de la résolution de ces problèmes est assuré.

Simplifications. Une pIMC est dite convexe si et seulement si, pour tout état s, il existe un chemin

2. Dans le cas de solveur *complets*. Cependant, afin d'améliorer la performance des solveurs, il est aussi possible d'utiliser des solveurs *incomplets* lorsqu'il n'est pas nécéssaire de prouver la non-satisfiabilité.

de l'état initial vers s empruntant uniquement des transitions non réduites à $[0, 0]$. Au vu des problèmes considérés, nous faisons l'hypothèse, en toute généralité, que toutes les pIMCs que nous manipulons sont convexes. De plus, nous limitons l'ensemble $\mathsf{Int}_{[0,1]}(X)$ aux intervales fermés. Néanmoins, toutes nos modélisations peuvent être aisément étendues à des intervalles ouverts/semi-ouverts dont les bornes sont des combinaisons linéaires de paramètres et de constantes.

3 Consistance Existentielle

La première propriété fondamentale des pIMCs que nous considérons est la *consistance existentielle*. Une pIMC satisfait cette propriété si et seulement si il existe au moins une MC concrète $\mathcal{M}$ qui la satisfait. On dira alors que $\mathcal{P}$ est *consistante*. En pratique, $\mathcal{P}$ est consistante si et seulement si il existe une IMC $\mathcal{I}$ et une MC $\mathcal{M}$ telles que $\mathcal{M}$ satisfait $\mathcal{I}$ et $\mathcal{I}$ est une instance de $\mathcal{P}$.

Problem 1. Soit $\mathcal{P}$ une pIMC. $\mathcal{P}$ est-elle consistante ?

Dans [8], les auteurs ont prouvé (Theorème 1 ci-dessous) que le problème de la consistance pour les pIMCs peut se réduire en toute généralité à la recherche d'une MC concrète *ayant la même structure que la* pIMC *originale*, plutôt qu'une structure quelconque. Dans le même article, les auteurs ont proposé une modélisation du problème en un CSP de taille exponentielle dans la taille de la pIMC originale (*i.e.*, qui énumère dans le pire cas des contraintes sur tous les sous-ensembles d'états de la pIMC). Etant donné que notre modélisation diffère profondément de la leur, nous ne la rappellerons pas ici.

Theorem 1 ([8]). *Une* pIMC (S, s_0, P, A, V, Y) *est consistante si et seulement si il existe une implémentation de la forme* $\mathcal{M} = (S, s_0, p, A, V)$ *telle que, pour tout état* s *atteignable dans* $\mathcal{M}$, $p(s, s') \in P(s, s')$ *pour tout* $s' \in S$.

3.1 Modélisation du Problème 1

Nous proposons maintenant une modélisation CSP pour la résolution du problème de la consistance existentielle des pIMCs.

Formellement, étant donné une pIMC $\mathcal{P} = (S, s_0, P, A, V, Y)$, la modélisation $\mathbf{M}_{\exists\mathbf{c}}(\mathcal{P})$ que nous proposons est la suivante. Nous introduisons une variable π_p ayant pour domaine $[0, 1]$ pour chaque paramètre $p \in Y$, une variable $\theta_s^{s'}$ ayant pour domaine $[0, 1]$ pour chaque transition $(s, s') \in \{\{s\} \times \mathtt{Succ}(s) \mid s \in S\}$, et finalement une variable Booléenne ρ_s pour chaque état $s \in S$. Le but de chacune des variables booléennes ρ_s est d'indiquer si l'état s est atteignable dans la MC représentée par le modèle courant étant données les probabilités de transition définies pas les variables $\theta_s^{s'}$. Les contraintes sont définies comme suit :

(1) ρ_{s_0}

(2) $\neg\rho_s \Leftrightarrow \Sigma_{s' \in \mathtt{Pred}(s)\setminus\{s\}} \theta_{s'}^{s} = 0$, si $s \neq s_0$

(3) $\neg\rho_s \Leftrightarrow \Sigma_{s' \in \mathtt{Succ}(s)} \theta_s^{s'} = 0$

(4) $\rho_s \Leftrightarrow \Sigma_{s' \in \mathtt{Succ}(s)} \theta_s^{s'} = 1$

(5) $\bigwedge_{s' \in \mathtt{Succ}(s)} \rho_s \Rightarrow \theta_s^{s'} \in P(s, s')$

La contrainte (1) impose que l'état initial doit être atteignable. La contrainte (2) assure ensuite que pour tout état non-initial s, ρ_s vaut 0 si et seulement si s n'est atteignable depuis aucun état prédécesseur dans le graphe sous-jacent. La contrainte (3) propage la non-atteignabilité au successeurs, *i.e.*, toutes les probabilités des transitions sortantes d'un état non-atteignable sont fixées à 0. La contrainte (4) guarantit que les probabilités affectées aux transitions sortantes de tout état atteignable forment des distributions de probabilité correctes. Finalement, la contrainte (5) s'assure que les valeurs affectées aux transitions sortantes des états atteignables sont choisies correctement dans les intervalles spécifiés dans la pIMC.

Example 8. Soit $\mathcal{P}$ la pIMC donnée dans la Figure 4. La figure 5 illustre les variables introduites dans le modèle $\mathbf{M}_{\exists\mathbf{c}}(\mathcal{P})$: une variable pour chaque transition (*e.g.*, θ_0^1, θ_0^2, θ_1^1), une variable booléenne pour chaque état (*e.g.*, ρ_0, ρ_1), et une variable par paramètre (π_p et π_q).

Les contraintes présentées ci-après correspondent à l'instanciation des contraintes (2), (3), (4), et (5) de la modélisation $\mathbf{M}_{\exists\mathbf{c}}$ appliquée à l'état 2 de $\mathcal{P}$:

$\neg\rho_2 \Leftrightarrow \theta_0^2 = 0$ $\qquad$ $\rho_2 \Rightarrow \theta_2^1 \in [0, \pi_p]$

$\neg\rho_2 \Leftrightarrow \theta_2^1 + \theta_2^2 + \theta_2^4 = 0$ $\qquad$ $\rho_2 \Rightarrow \theta_2^2 \in [0.4, \pi_p]$

$\rho_2 \Leftrightarrow \theta_2^1 + \theta_2^2 + \theta_2^4 = 1$ $\qquad$ $\rho_2 \Rightarrow \theta_2^4 \in [0, 0.5])$

Finalement, une solution au CSP $\mathbf{M}_{\exists\mathbf{c}}(\mathcal{P})$ est donnée en Figure 6. Notons que la fonction de transition dérivée d'une solution au problème de la consistance d'une pIMC modélisé par $\mathbf{M}_{\exists\mathbf{c}}$ n'est pas forcément correcte. En effet $\mathbf{M}_{\exists\mathbf{c}}$ impose que les probabilités données aux transitions sortantes des états non-atteignables sont réduites à 0, ce qui ne permet pas de définir des distributions de probabilités correctes. Par exemple, la fonction de transition correspondant à la solution proposée dans la Figure 6 n'est pas correcte puisque les probabilités des transitions sortantes de l'état 4 ne forment pas une distribution de probabilité correcte (*i.e.*, $\theta_4^4 + \theta_4^3 \neq 1$). Néanmoins, étant donné que les états non-atteignables sont correctement identifiés et que leurs transitions sortantes n'ont aucun impact sur la consistance des pIMCs considérées, ce n'est pas

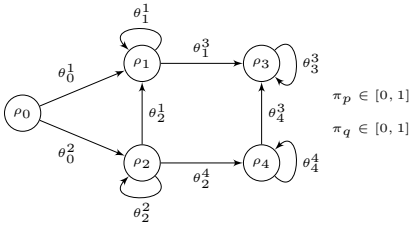


FIGURE 5 – Variables pour la résolution de la consistance existentielle pour la pIMC $\mathcal{P}$ donnée en Fig. 4

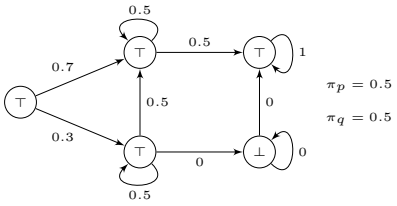


FIGURE 6 – Une solution du CSP $\mathbf{M}_{\exists \mathbf{c}}(\mathcal{P})$ pour la pIMC $\mathcal{P}$ de la Fig. 4

vraiment problématique. Deux solutions triviales sont alors possibles : (1) retirer les états non-atteignables des MCs produites, ou (2) assigner une distribution de probabilité arbitraire à leur transitions sortantes.

Proposition 1. *Soit $\mathcal{P}$ une* pIMC*, le* CSP $\mathbf{M}_{\exists \mathbf{c}}(\mathcal{P})$ *est satisfiable si et seulement si la* pIMC $\mathcal{P}$ *est consistente.*

Démonstration. Soit $\mathcal{P} = (S, s_0, P, A, V, Y)$ une pIMC.

[$\Rightarrow$] Si le CSP $\mathbf{M}_{\exists \mathbf{c}}(\mathcal{P}) = (X, D, C)$ est satisfiable, alors il existe une valuation v des variables de X qui satisfait toutes les contraintes de C.

Définissons la MC $\mathcal{M} = (S, s_0, p, A, V)$ telle que $p(s, s') = v(\theta_s^{s'})$ si $\theta_s^{s'} \in \Theta$ et $p(s, s') = 0$ sinon.

Dans un premier temps, nous montrons par induction que pour tout état $s \in S$, "s est atteignable dans $\mathcal{M}$ si et seulement si $v(\rho_s) = 1$". D'après la contrainte (1), l'état initial satisfait triviallement cette propriété. Soit $s \in S$ un état quelconque et supposons que la propriété est satisfaite par tous ses prédecesseurs stricts (différents de lui-même). D'après la contrainte (2), $v(\rho_s) = 1$ si et seulement si il existe au moins un prédécesseur $s'' \neq s$ qui atteint s avec une probabilité non nulle (*i.e.*, $v(\theta_{s''}^s) \neq 0$). D'après la contrainte (3), ceci est uniquement possible si $v(\rho_{s''}) = 1$. En conséquence, $v(\rho_s) = 1$ si et seulement si il existe un état atteignable s'' t.q. $v(\theta_{s''}^s) > 0$.

Dans un second temps, nous prouvons que $\mathcal{M}$ satisfait la pIMC $\mathcal{P}$. Pour ce faire, nous utilisons le théorème 1. Nous avons montré ci-dessus que $v(\rho_s) = 1$ pour tout état s atteignable dans $\mathcal{M}$. La contrainte (5) implique donc que pour tout état atteignable s dans $\mathcal{M}$, on a $p(s, s') \in P(s, s')$ pour tout s'. La MC $\mathcal{M}$ ci-dessus satisfait alors triviallement la pIMC $\mathcal{P}$.

[$\Leftarrow$] Si la pIMC $\mathcal{P}$ est consistante, alors le théorème 1 guarantit qu'il existe une implémentation de la forme $\mathcal{M} = (S, s_0, p, A, V)$ dans laquelle, pour tout état atteignable s, on a $p(s, s') \in P(s, s')$ pour tout s' in S. De plus, il existe une IMC $\mathcal{I}$ et une valuation des paramètres $v^{\mathcal{I}}$ telle que $\mathcal{M}$ satisfait $\mathcal{I}$ et $\mathcal{I} = v^{\mathcal{I}}(\mathcal{P})$.

Soit $\mathcal{M}' = (S, s_0, p', A, V)$ la MC telle que, pour tout état non-atteignable $s \in S$, $p'(s, s') = 0$ for all $s' \in S$. Soit v une valuation telle que $v(\rho_s) = 1$ si et seulement si s est atteignable dans $\mathcal{M}$, $v(\theta_s^{s'}) = p'(s, s')$ et telle que v correspond) $v^{\mathcal{I}}$ pour les paramètres $y \in Y$. Par construction, v satisfait alors le CSP $\mathbf{M}_{\exists \mathbf{c}}(\mathcal{P})$. □

La modélisation proposée ici résout le même problème que celle de [9] (la consistance existentielle d'une pIMC), en utilisant un CSP de taille linéaire en la taille de la pIMC d'origine alors que celle de [9] est exponentielle. Les deux modélisations utilisent des inégalités linéaires et des contraintes logiques, mais la solution proposée ici est plus facile à comprendre (pas de récursion profonde lors de la construction des modèles, contrairement à ce qui est proposé dans [9]).

4 Problèmes d'atteignabilité

Nous nous intéressons maintenant à l'ensemble des MCs concrètes qui satisfont une pIMC donnée dans lesquelles un ensemble donné d'états est atteignable, *i.e.*, au moins un état de cet ensemble est atteignable. C'est ce que nous appelons le problème d'atteignabilité.

Nous allons bénéficier ici de la modularité de la programmation par contrainte. En effet, étant donné que l'ensemble des MCs qui satisfont le problème d'atteignabilité est un sous-ensemble des solutions au problème de la consistance existentielle pour la même pIMC, nous allons simplement compléter notre modélisation $\mathbf{M}_{\exists \mathbf{c}}$ en y ajoutant des contraintes bien choisies pour résoudre l'atteignabilité.

Soit $\mathcal{P} = (S, s_0, P, A, V, Y)$ une pIMC et $G \subseteq S$ un ensemble d'états cible. Deux problèmes d'atteignabilité peuvent être définis : l'*atteignabilité existentielle* et l'*atteignabilité universelle*. Nous commençons par le premier et le second sera traîté dans la section suivante. Intuitivement, G est *atteignable existentiel* (ou seulement atteignable) dans $\mathcal{P}$ si et seulement si *il existe* une MC $\mathcal{M}$ qui satisfait $\mathcal{P}$ dans laquelle G est atteignable avec une probabilité strictement positive (*i.e.*, dans laquelle il existe un état $g \in G$ qui est atteignable depuis l'état initial avec une probabilité strictement positive).

Problem 2. Soit $\mathcal{P} = (S, s_0, P, A, V, Y)$ une pIMC et $G \subseteq S$ un ensemble d'états cible. G est-il atteignable dans $\mathcal{P}$ avec une probabilité strictement positive ?

4.1 Modélisation du Problème 2

Soient $\mathcal{P} = (S, s_0, P, A, V, Y)$ une pIMC et $G \subseteq S$ un ensemble d'états cible. Nous contruisons la modélisation $\mathbf{M}_{\exists\mathbf{r}}$ pour vérifier l'atteignabilité existentielle de G dans $\mathcal{P}$ en complétant la modélisation $\mathbf{M}_{\exists\mathbf{c}}$. Formellement, $\mathbf{M}_{\exists\mathbf{r}}(\mathcal{P}) = (X \cup X', D \cup D', C \cup C')$ est tel que $(X, D, C) = \mathbf{M}_{\exists\mathbf{c}}(\mathcal{P})$, X' contient une variable entière ω_s de domaine $[0, |S|]$ par état s dans S, D' contient les domaines de ces variables et C' est composé des contraintes suivantes pour chaque état $s \in S$:

(1) $\omega_s = 1$, si $s = s_0$

(2) $\omega_s \neq 1$, si $s \neq s_0$

(3) $\rho_s \Leftrightarrow (\omega_s \neq 0)$

(4) $\omega_s > 1 \Rightarrow \bigvee_{s' \in \mathtt{Pred}(s) \setminus \{s\}} (\omega_s = \omega_{s'} + 1) \wedge (\theta_s^{s'} > 0)$, si $s \neq s_0$

(5) $\omega_s = 0 \Leftrightarrow \bigwedge_{s' \in \mathtt{Pred}(s) \setminus \{s\}} (\omega_{s'} = 0) \vee (\theta_s^{s'} = 0)$, si $s \neq s_0$

Rappelons en premier lieu que le CSP $\mathbf{M}_{\exists\mathbf{c}}(P)$ construit une chaîne de Markov $\mathcal{M}$ qui satisfait $\mathcal{P}$. Ainsi pour chaque état s de $\mathcal{M}$ l'ajout des contraintes (1), (2), (4) et (5) dans $\mathbf{M}_{\exists\mathbf{r}}$ assure que : $\omega_s = k$ ssi il existe dans $\mathcal{M}$ un chemin depuis l'état initial jusque s de taille $k-1$ ayant une probabilité non nulle ; et l'état s n'est pas atteignable dans $\mathcal{M}$ à partir de l'état initial s_0 ssi ω_s vaut 0. Enfin, la constrainte (3) force la variable booléenne ρ_s indicatrice d'atteignabilité à valoir true ssi il existe dans $\mathcal{M}$ un chemin partant de l'état initial s_0 et atteignant s ayant une probabilité non nulle (*i.e.*, $\omega_s \neq 0$).

Soit G un ensemble d'états dans $\mathcal{P}$. Nous proposons le CSP $\mathbf{M}_{\exists\mathbf{r}}(\mathcal{P}, G)$ qui étend $\mathbf{M}_{\exists\mathbf{r}}(\mathcal{P})$ comme réponse au problème 2. Nous considérons exactement les mêmes variables avec les mêmes domaines, et nous gardons également les contraintes de $\mathbf{M}_{\exists\mathbf{r}}(\mathcal{P})$. Formellement, $\mathbf{M}_{\exists\mathbf{r}}(\mathcal{P}, G) = (X, D, C \cup C')$ est tel que $\mathbf{M}_{\exists\mathbf{r}}(\mathcal{P}) = (X, D, C)$ et $C' = \bigvee_{s \in G} \rho_s$.

Proposition 2. *Soient $\mathcal{P} = (S, s_0, P, A, V, Y)$ une pIMC et $G \subseteq S$ un ensemble d'états cible. Le CSP $\mathbf{M}_{\exists\mathbf{r}}(\mathcal{P}, G)$ est satisfiable si et seulement si G est atteignable existentiel dans $\mathcal{P}$.*

Comme pour le Problem 1, nous avons un gain exponentiel en terme de la taille de la modélisation comparée à la modélisation état de l'art de [9] : le nombre de contraintes et variables dans $\mathbf{M}_{\exists\mathbf{r}}$ est linéaire en terme de la tailles des modèles des pIMC. Le problème suivant (et le dernier) résolu dans la littérature pour les pIMCs, est le problème d'atteignabilité universelle. Alors que le problème de consistance existentielle cherche une MC concrète avec un état (ou ensemble d'états) donné atteignable, l'atteignabilité universelle vérifie cette propriété pour toutes les MCs concrètes satisfaisant la pIMC. Soient $\mathcal{P} = (S, s_0, P, A, V)$ une pIMC et $G \subseteq S$, nous disons que G est *atteignable universel* dans $\mathcal{P}$ ssi quelque soit l'implementation $\mathcal{M}$ de $\mathcal{P}$, G est atteignable dans $\mathcal{M}$.

Problem 3. Soient $\mathcal{P} = (S, s_0, P, A, V, Y)$ une pIMC et $G \subseteq S$ un ensemble d'états cible. G est-il atteignable universel dans $\mathcal{P}$?

4.2 Modélisation du Problème 3

Soient $\mathcal{P} = (S, s_0, P, A, V, Y)$ une pIMC et $G \subseteq S$ un ensemble d'états cible, nous construisons la modélisation $\mathbf{M}_{\forall\mathbf{r}}$ pour vérifier le problème d'atteignabilité universelle de façon duale au problème d'atteignabilité existentielle. La modélisation $\mathbf{M}_{\forall\mathbf{r}}$ étend la modélisation $\mathbf{M}_{\exists\mathbf{r}}$. Formellement, $\mathbf{M}_{\forall\mathbf{r}}(\mathcal{P}, G) = (X, D, C \cup C')$ t.q. $\mathbf{M}_{\exists\mathbf{r}}(\mathcal{P}) = (X, D, C)$ et $C' = \bigwedge_{s \in G} \neg\rho_s$.

Proposition 3. *Soient $\mathcal{P} = (S, s_0, P, A, V, Y)$ une pIMC et $G \subseteq S$ un ensemble d'états cible. $\mathbf{M}_{\forall\mathbf{r}}(\mathcal{P}, G)$ est insatisfiable ssi G est atteignable universel dans $\mathcal{P}$.*

Démonstration. Soient $\mathcal{P} = (S, s_0, P, A, V, Y)$ une pIMC et $G \subseteq S$ un ensemble d'états cible. Le CSP $\mathbf{M}_{\forall\mathbf{r}}(\mathcal{P}, G) = (X, D, C \cup C')$ t.q. $\mathbf{M}_{\exists\mathbf{r}}(\mathcal{P}) = (X, D, C)$ et $C' = \bigvee_{s \in G} \neg\rho_s$ est insatisfiable est équivalent à : quelque soit la valuation v, $v(C)$ est faux ou $v(C')$ est faux. Grâce à la proposition 2, ceci est équivalent à : quelque soit la MC $\mathcal{M}$, $\mathcal{M}$ ne satisfait pas $\mathcal{P}$ ou "tous les états but g de G ne sont pas atteignables dans $\mathcal{M}$" est faux. Pour conclure, ceci est équivalent à : $\mathcal{M}$ ne satisfait pas $\mathcal{P}$ ou il existe un état but g de G atteignable dans $\mathcal{M}$. □

Nous avons encore le même gain exponentiel entre les modèles produits par notre modélisation $\mathbf{M}_{\forall\mathbf{r}}$ et les modèles état de l'art produits à partir de [9]. Nous proposons une implémentation des 2 approches et nous présentons des expérimentations pour résoudre le problème de la consistance existentielle.

5 Expérimentations

Les modélisation état de l'art de [9] (**SotA** en court) n'ont jamais été implémentées. En particulier, il n'y avait pas de réel ensemble de benchmarks pour des evaluations expérimentales sur les pIMCs. Dans cette section, nous présentons d'abord une méthode pour générer une grande variété de pIMCs hétérogènes basée sur des benchmarks de la communauté MC [14]. Ensuite, nous présentons l'implémentation du problème

de la consistance existentielle basée sur les modélisations **SotA** et $\mathbf{M}_{\exists\mathbf{c}}$. Finallement, nous implémentons nos benchmarks pour 2 méthodes de résolutions différentes, Satisfiability Modulo Theory et Mixed Integer Linear Programming, et nous comparons les 2 implémentations. [3]

5.1 Benchmarks

Les MCs ont été utilisées pendant des décennies pour modéliser des applications réelles. PRISM [14] est une référence pour la vérification de systèmes probabilistes temps-réel. En particulier, il est capable de vérifier des propriétés des MCs. Comme dit en Section 2, les pIMCs correspondent à des abstractions de MCs. PRISM référencie de nombreux benchmarks basés sur les MCs [4]. Nous construisons des pIMCs à partir de MCs en remplaçant aléatoirement des probabilités exactes sur des transitions par des intervalles (paramétrés) de probabilités. Notre générateur de pIMCs nécessite 4 arguments : la fonction de transition de la MC ; le nombre de paramètres pour la pIMC générée ; le rapport nombre d'intervalles sur nombre de transitions dans la pIMC générée ; le rapport nombre de paramètres sur nombre de bornes d'intervalles pour la pIMC générée. [5]

L'ensemble de nos benchmarks est présenté dans le tableau 1. Nous avons sélectionné 5 applications : HERMAN - le "self-stabilisation protocol" d'Herman [15] ; EGL - le "contract signing protocol" d'Even, Goldreich & Lempel [17] ; BRP - le "bounded retransmission protocol" de [6] ; CROWDS - le "crowds protocol" de [19] ; et NAND - le "nand multiplexing" de [16]. Chacun est instancié en définissant des constantes globales (*e.g.*, N pour l'application HERMAN, L et N pour EGL) conduisant ainsi à de plus ou moins complexes MCs. Avec notre générateur de pIMCs, nous avons généré un ensemble de benchmarks hétérogènes : 459 pIMCs avec de 8 à 15 102 états et de 28 à 21 567 transitions non réduites à $[0, 0]$. Les pIMCs contiennent de 2 à 250 paramètres sur 0 à 7 772 intervalles.

5.2 Modélisation

Ni les modélisations **SotA**, ni les modélisations $\mathbf{M}_{\exists\mathbf{c}}$ n'avaient été implémentées. La modélisation **SotA** nécessite des variables continues contraintes par des inéquations linéaires strictes et non-strictes, et des constraintes logiques (conjonctions et disjonctions imbriquées). La modélisation $\mathbf{M}_{\exists\mathbf{c}}$ repose sur des inéquations linéaires non strictes et des contraintes logiques (implications, conjonctions et disjunctions). Les 2 modélisations peuvent être formulées dans la logique QF_LRA afin d'être résolues par un solveur "Satisfiability Modulo Theory" (SMT) [2]. La logique QF_LRA concerne les combinaisons booléennes d'inéquations entre des polynômes linéaires sur des variables réelles. Cependant, si le but n'est pas d'obtenir des solutions exactes sur les réels, la modélisation peut se faire en "Mixed Integer Linear Programs" (MILP) [20]. Le modeleur a été implémenté en Python.

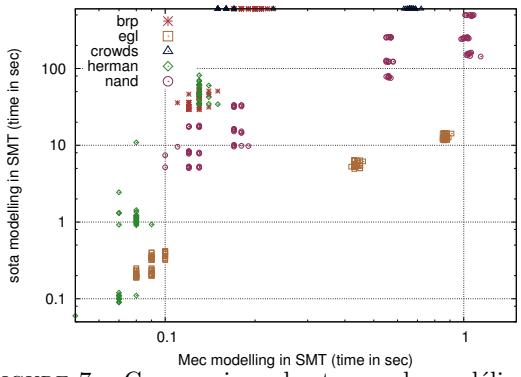


FIGURE 7 – Comparaison des temps de modélisation pour le problème de consistance existentielle

Nous n'évaluons aucune expression arithmétique lors de la modélisation. De la même façon, les valeurs aux bornes des intervalles des pIMCs sont lues comme des chaînes et aucune simplification, même triviale n'est faite. Le but est d'éviter les arrondis aux bornes des intervalles lors de l'utilisation de nombres flottants. [6]

Les expérimentations ont été réalisées sur un processeur Intel Core i5 à 2.4 GHz. Le time-out a été fixé à 10 minutes et le memory-out à 2Gb. Le tableau 2 présente la taille des instances (*i.e.*, nombre de variables et nombre de contraintes) pour résoudre le problème de consistance existentielle de nos benchmarks en utilisant (1) la modélisation SMT **SotA**, (2) la modélisation SMT $\mathbf{M}_{\exists\mathbf{c}}$, et (3) la modélisation MILP $\mathbf{M}_{\exists\mathbf{c}}$. Notons d'abord que tous les pIMCs compilent avec notre modélisation $\mathbf{M}_{\exists\mathbf{c}}$ alors que la modélisation **SotA** produit des "out of memory errors" pour 4 ensembles de benchmarks : plus de 20% des instances (voir cellules OM dans le tableau 2). Rappelons que la modélisation **SotA** est définie inductivement et qu'elle itère sur l'ensemble des parties des états. En pratique, cela implique de profondes récursions liées à l'énumération sur l'ensemble des parties des états. Le gain exponentiel décrit en section 3 est visible en terme du nombre

3. Toutes les ressources, benchmarks, et code source Python sont disponibles en ligne à https://github.com/anicet-bart/pimc_pylib

4. Voir la catégorie chaînes de Markov à temps discret sur le website de PRISM

5. Voir générateur de pIMC à https://github.com/anicet-bart/pimc_pylib

6. Voir modeleur pIMC à https://github.com/anicet-bart/pimc_pylib

Ens. de benchmarks		#pIMCs	#états	#transitions	#intervalles min	#intervalles moy	#intervalles max	#paramAuxBornes min	#paramAuxBornes moy	#paramAuxBornes max	#paramètres
HERMAN	N=3	27	8	28	0	7	18	0	3	11	{2, 5, 10}
HERMAN	N=5	27	32	244	19	50	87	0	12	38	{2, 5, 10}
HERMAN	N=7	27	128	2188	37	131	236	3	31	74	{5, 15, 30}
EGL	L=2; N=2	27	238	253	16	67	134	0	15	57	{2, 5, 10}
EGL	L=2; N=4	27	6910	7165	696	1897	3619	55	444	1405	{2, 5, 10}
EGL	L=4; N=2	27	494	509	47	136	276	3	32	115	{2, 5, 10}
EGL	L=4; N=4	27	15102	15357	1448	4068	7772	156	951	3048	{2, 5, 10}
BRP	M=3; N=16	27	886	1155	16	64	135	1	15	45	{2, 5, 10}
BRP	M=3; N=32	27	1766	2307	40	128	275	3	32	129	{2, 5, 10}
BRP	M=4; N=16	27	1095	1443	22	80	171	0	20	62	{2, 5, 10}
BRP	M=4; N=32	27	2183	2883	49	164	323	3	39	139	{2, 5, 10}
CROWDS	CS=10; TR=3	27	6563	15143	1466	3036	4598	57	235	535	{5, 15, 30}
CROWDS	CS=5; TR=3	27	1198	2038	190	410	652	8	31	76	{5, 15, 30}
NAND	K=1; N=10	27	7392	11207	497	980	1416	109	466	1126	{50, 100, 250}
NAND	K=1; N=5	27	930	1371	60	121	183	9	58	159	{50, 100, 250}
NAND	K=2; N=10	27	14322	21567	992	1863	2652	197	866	2061	{50, 100, 250}
NAND	K=2; N=5	27	1728	2505	114	217	329	23	101	263	{50, 100, 250}

TABLE 1 – Benchmarks composés de 459 pIMCs sur 5 familles

Ens. de benchmarks		#variables (1)	#variables (2)	#variables (3)	#contraintes (1)	#contraintes (2)	#contraintes (3)	moy(temps modèlisation) (1)	moy(temps modèlisation) (2)	moy(temps modèlisation) (3)	moy(temps résolution) (1)	moy(temps résolution) (2)	moy(temps résolution) (3)
HERMAN	N=3	71	42	42	1258	272	238	0.10	0.07	0.07	0.01	0.01	0.01
HERMAN	N=5	1,031	282	282	51,064	2,000	1,750	1.52	0.08	0.08	0.24	0.03	0.01
HERMAN	N=7	16,402	2,333	2,333	1,293,907	16,483	14,278	50.47	0.13	0.13	5.92	0.28	0.04
EGL	L=2; N=2	462	497	497	4,609	3,917	3,658	0.21	0.08	0.08	0.02	0.04	0.01
EGL	L=2; N=4	13,786	14,081	14,081	138,596	112,349	105,178	5.66	0.44	0.44	0.54	2.15	0.36
EGL	L=4; N=2	958	1,009	1,009	9,560	8,013	7,498	0.36	0.10	0.10	0.04	0.08	0.02
EGL	L=4; N=4	30,138	30,465	30,465	301,866	243,421	228,058	13.03	0.87	0.87	1.26	11.31	0.97
BRP	MAX=3; N=16	68,995	2,047	2,047	738,580	16,063	14,902	32.29	0.12	0.12	3.54	0.21	0.06
BRP	MAX=3; N=32	OM	4,079	4,079	OM	32,047	29,734	OM	0.18	0.18	OM	0.47	0.13
BRP	MAX=4; N=16	103,105	2,544	2,544	1,114,774	19,960	18,511	46.54	0.13	0.13	5.42	0.27	0.08
BRP	MAX=4; N=32	OM	5,072	5,072	OM	39,832	36,943	OM	0.21	0.21	OM	0.63	0.17
CROWDS	CS=10; TR=3	OM	21,723	21,723	OM	165,083	149,923	OM	0.67	0.66	OM	11.48	0.79
CROWDS	CS=5; TR=3	OM	3,253	3,253	OM	25,063	23,008	OM	0.16	0.15	OM	0.39	0.09
NAND	K=1; N=10	87,506	18,732	18,732	888,733	145,108	133,768	152.06	0.56	0.56	3.72	6.21	0.79
NAND	K=1; N=5	6,277	2,434	2,434	62,987	18,098	16,594	10.26	0.12	0.12	0.24	0.25	0.07
NAND	K=2; N=10	169,786	36,022	36,022	1,722,970	279,998	258,298	298.93	1.04	1.04	7.75	31.81	2.06
NAND	K=2; N=5	11,623	4,366	4,366	117,814	33,218	30,580	19.24	0.17	0.17	0.44	0.48	0.13

TABLE 2 – Comparaisons des tailles, temps de modélisation et de résolution pour 3 approches : (1) la modélisation State-of-the-Art en SMT, (2) notre modélisation en SMT, et (3) notre modélisation en MILP (temps en secondes).

de variables et contraintes (tableau 2), et en terme de temps de modélisation (figure 7). Chaque point de la figure 7 correspond à une instance de nos benchmarks. Alors que le temps de modélisation varie de 0 à 1s. avec la modélisation $\mathbf{M}_{\exists\mathbf{c}}$, il varie de 0 à 500s. avec la modélisation **SotA** (s'il n'y a pas de "out of memory error").

La formulation MILP des contraintes logiques (*e.g.*, conjonction, disjonction, implication, équivalence) implique l'introduction de variables binaires appelées "indicator variables" [3]. Chaque "indicator variable" est associée à une ou plusieurs contraintes. La valuation des "indicator variables" active ou désactivate les contraintes associées. Nous avons tenté de formuler la modélisation **SotA** en MILP. Les conjonctions et disjonctions imbriquées impliquent l'introduction de nombreuses "indicator variables", ce qui conduit à de gigantesques instances et en conséquence, à des très mauvais temps de modélisation et résolution. Cependant, puisque les variables booléennes dans $\mathbf{M}_{\exists\mathbf{c}}$ correspondent exactement aux "indicator variables", la formulation MILP de la modélisation $\mathbf{M}_{\exists\mathbf{c}}$ n'introduit pas de variables additionnelles, ni de contraintes. La différence entre $\mathbf{M}_{\exists\mathbf{c}}$ en SMT et $\mathbf{M}_{\exists\mathbf{c}}$ en MILP vient de l'encodage des domaines des variables continues : en SMT, il faut des contraintes d'inéquations, *e.g.*, $0 \leq x \leq 1$, pour x une variable d'un CSP. Le temps

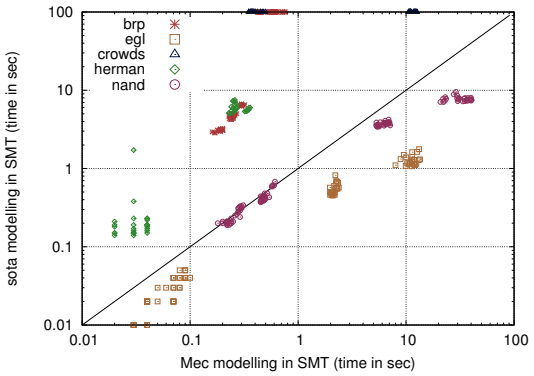


FIGURE 8 – Comparaison des temps de résolution pour le problème de consistance existentielle

de modélisation est le même pour SMT et MILP pour le modèle $\mathbf{M}_{\exists\mathbf{c}}$.

5.3 Résolution

Nous avons choisi le solveur SMT Z3 [7] (version v. 4.4.2) et le solveurs MILP CPLEX [1] (version v. 12.6.3.0). Les solveurs n'ont pas été tunés et nous avons utilisé leurs stratégies par défaut. Comme auparavant, les expérimentations ont été réalisées sur un processeur Intel Core i5 à 2.4 GHz, avec un time-out de

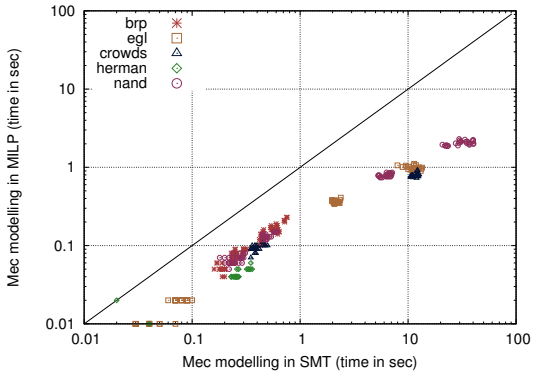


FIGURE 9 – Comparaison des temps de résolution entre les formulations SMT et MILP

10 minutes. Le tableau 2 présente les temps de résolution pour la consistance ielle sur nos benchmarks avec (1) la modélisation SMT **SotA**, (2) la modélisation SMT $\mathbf{M}_{\exists \mathbf{c}}$, et (3) la modélisation MILP $\mathbf{M}_{\exists \mathbf{c}}$. Alors que les modèles **SotA** sont plus gros que les modèles $\mathbf{M}_{\exists \mathbf{c}}$, le temps de résolution pour les modèles **SotA** est compétitif avec le temps de résolution pour les modèles $\mathbf{M}_{\exists \mathbf{c}}$. Le *scatter plot* figure 8 (échelle logarithmique) compare les temps de résolution pour les modélisations SMT **SotA** et les modélisations SMT $\mathbf{M}_{\exists \mathbf{c}}$. Cependant, si l'on considère le temps total de résolution du problème (*i.e.*, temps de modélisation plus temps de résolution) la modélisation $\mathbf{M}_{\exists \mathbf{c}}$ est clairement plus rapide que la modélisation **SotA**. Finalement, la comparaison des temps de résolution des modélisations SMT $\mathbf{M}_{\exists \mathbf{c}}$ et MILP $\mathbf{M}_{\exists \mathbf{c}}$ est donnée figure 9. Cela montre que la perte de sécurité impliquée par le passage des réels de Z3 (SMT) aux flottants de CPLEX (MILP) conduit à des gains non négligeables en terme de temps de résolution (pratiquement un gain exponentiel sur nos benchmarks). Dans les faits, la modélisation SMT $\mathbf{M}_{\exists \mathbf{c}}$ requiert 50s pour la résolution alors que celle MILP $\mathbf{M}_{\exists \mathbf{c}}$ nécessite moins de 5s pour les mêmes instances.

6 Conclusion

Dans ce travail, nous avons nettement amélioré (en temps et en espace) la modélisation état de l'art présentée dans [9] pour la vérification de 3 propriétés fondamentales des pIMCs : les problèmes de consistance existentielle, d'atteignabilité existentielle et d'atteignabilité universelle. Nous avons d'abord exposé nos modélisations $\mathbf{M}_{\exists \mathbf{c}}$, $\mathbf{M}_{\exists \mathbf{r}}$ et $\mathbf{M}_{\forall \mathbf{r}}$ pour résoudre ces problèmes. Nos modèles sont de taille linéaire en la taille des pIMCs alors que les modèles état de l'art sont exponentiels en la taille des pIMCs. Nous avons ensuite réalisé des expérimentations sur ces modélisations : nous avons présenté une méthode pour générer des pIMCs à partir de MCs et avons construit un vaste ensemble de benchmarks hétérogènes ; nous avons développé un modeleur pour le problème de consistance existentielle pour la modélisation état de l'art et notre modélisation $\mathbf{M}_{\exists \mathbf{c}}$ en SMT et MILP. Les résultats montrent que notre modélisation accroît grandement les performances en temps et espace pour résoudre ces problèmes. Dans le futur, nous prévoyons de vérifier d'autres propriétés des pIMCs.

Références

[1] IBM ILOG CPLEX Optimizer, Last 2010.

[2] Clark W Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. Satisfiability modulo theories. *Handbook of satisfiability*, 185 :825–885, 2009.

[3] Pietro Belotti, Pierre Bonami, Matteo Fischetti, Andrea Lodi, Michele Monaci, Amaya Nogales-Gómez, and Domenico Salvagnin. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, pages 1–22, 2016.

[4] Michael Benedikt, Rastislav Lenhardt, and James Worrell. LTL model checking of interval markov chains. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013*, pages 32–46, 2013.

[5] Krishnendu Chatterjee, Koushik Sen, and Thomas A. Henzinger. Model-checking omega-regular properties of interval markov chains. In *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008*, pages 302–317, 2008.

[6] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In L. de Alfaro and S. Gilmore, editors, *Proc. 1st Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modelling and Verification (PAPM/PROBMIV'01)*, 2001.

[7] Leonardo De Moura and Nikolaj Bjørner. Z3 : An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] Benoît Delahaye. Consistency for parametric interval markov chains. In *2nd International Workshop on Synthesis of Complex Parameters, SynCoP 2015, April 11, 2015, London, United Kingdom*, pages 17–32, 2015.

[9] Benoît Delahaye, Didier Lime, and Laure Petrucci. Parameter synthesis for parametric interval markov chains. In *Verification, Model Checking, and Abstract Interpretation - 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings*, pages 372–390, 2016.

[10] Itzhak Gilboa, Andrew W. Postlewaite, and David Schmeidler. Probability and Uncertainty in Economic Modeling. *Journal of Economic Perspectives*, 22(3) :173–88, Summer 2008.

[11] Marco Gribaudo, Daniele Manini, and Anne Remke, editors. *Model Checking of Open Interval Markov Chains*, Cham, 2015. Springer International Publishing.

[12] Dirk Husmeier, Richard Dybowski, and Stephen Roberts. *Probabilistic Modeling in Bioinformatics and Medical Informatics*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[13] Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 266–277, 1991.

[14] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0 : Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[15] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic verification of herman's self-stabilisation algorithm. *Formal Aspects of Computing*, 24(4) :661–670, 2012.

[16] G. Norman, D. Parker, M. Kwiatkowska, and S. Shukla. Evaluating the reliability of NAND multiplexing with PRISM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(10) :1629–1637, 2005.

[17] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6) :561–589, 2006.

[18] Carles Roig, Lorenzo J. Tardón, Isabel Barbancho, and Ana M. Barbancho. Automatic melody composition based on a probabilistic model of music style and harmonic rules. *Knowledge-Based Systems*, 71 :419 – 434, 2014.

[19] V. Shmatikov. Probabilistic model checking of an anonymity system. *Journal of Computer Security*, 12(3/4) :355–377, 2004.

[20] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *SIAM Review*, 57(1) :3–57, 2015.

Stratégies de recherche pour les systèmes de contraintes sur les flottants *

Heytem Zitoun[1] †**Claude Michel**[1] **Michel Rueher**[1] **Laurent Michel**[2]

[1] Université Côte d'Azur, CNRS, I3S, France
[2] University of Connecticut, Storrs, CT 06269-2155
prenom.nom@unice.fr ldm@engr.uconn.edu

Résumé

La vérification logicielle est un enjeu clé pour les applications critiques telles que l'aviation, l'aérospatiale, ou les systèmes embarqués. Les approches basées sur le « bounded model checking (BMC) » et la programmation par contraintes (CBMC, CBPV, ...) recherchent des contre-exemples qui violent une propriété du programme. La recherche de tels contre-exemples peut-être particulièrement longue et couteuse lorsque le programme à vérifier contient des opérations sur les flottants. En effet, les stratégies de recherche existantes en programmation par contraintes ont été conçues pour les domaines finis et, dans une moindre mesure, les domaines continus. Dans cet article, nous proposons de nouvelles stratégies de recherches adaptées aux spécificités des flottants. Ces stratégies se basent sur des propriétés propres aux domaines des variables (e.g., densité des domaines) et aux contraintes sur les flottants (e.g., arithmétique propre aux flottants et structure même des problèmes). Les stratégies que nous introduisons exploitent ces propriétés et les combinent en utilisant, par exemple, la notion de score introduite dans la stratégie « Impact Based Search ». Pour évaluer ces nouvelles stratégies, nous avons porté le solveur sur les flottants FPCS dans Objective-CP afin de pouvoir bénéficier des facilités offertes par cet environnement.

1 Introduction

Un point important lors de la vérification de programme contenant des calculs sur les flottants est la recherche des erreurs de calcul dues à l'arithmétique spécifique des flottants qui peuvent entrainer un résultat sensiblement différent du résultat attendu sur les réels. Soit le programme foo suivant :

```
void foo(){
        float a = 1e8f;
        float b = 1.0f;
        float c = -1e8f;
        float r = a + b + c;
        if(r >= 1.0f){
                doThenPart();
        }else{
                doElsePart();
        }
}
```

En interprétant le programme foo sur les réels, l'instruction doThenPart devrait être exécutée. Cependant, un problème d'absorption sur les flottants (la valeur 1 est absorbée par 1e8f [1]) conduit le programme à passer par la branche Else.

Lors de la vérification de programmes avec des techniques de bounded model checking il faut aussi s'assurer qu'une propriété ne peut pas être violée du fait des erreurs de calcul sur les flottants. La programmation par contraintes [4, 5] a été utilisée pour traiter de tels problèmes, mais la recherche de contre-exemple reste longue et coûteuse. Les stratégies de recherche standards sont en effet peu efficaces pour trouver de tels contre-exemples. De nombreuses stratégies de recherche ont été proposées sur les entiers [3, 6, 12, 14, 15] et, dans une moindre mesure, sur les réels [10, 9]. Or, les stratégies propres aux entiers et aux réels sont mal adaptées aux flottants. Le sous-ensemble des entiers d'un domaine borné par deux entiers est fini et réparti de manière uniforme ; il est donc possible d'en énumérer toutes les valeurs lorsque

*Ces travaux ont été partiellement supportés par l'ANR COVERIF (ANR-15-CE25-0002).

†Papier doctorant : Heytem Zitoun[1] est auteur principal.

1. sur les flottants simple précision et avec un arrondi au plus près.

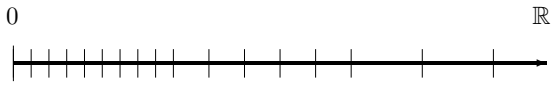


FIGURE 1 – Répartition des flottants

le domaine est suffisamment petit. Le sous-ensemble des réels borné par deux flottants étant infini et uniforme, il n'est pas énumérable. Aussi les stratégies de recherche adaptées au continu utilisent des techniques sur les intervalles telles que la bissection ainsi que des propriétés mathématiques pour prouver l'existence où l'absence de solution dans un petit intervalle. A contrario, l'ensemble des flottants est lui fini, mais sa cardinalité est très élevée et la répartition des flottants n'est pas du tout uniforme (la moitié des flottants sont dans le domaine [-1,1]). Les techniques précédentes, comme l'énumération, sont donc inenvisageables dans le cas général. Les flottants correspondent à une approximation des réels, mais n'hérite pas des mêmes propriétés mathématiques, telles que la continuité. Il est donc difficile de tirer profit des stratégies de recherches sur les réels comme celles qui exploite la continuité.

Dans cet article, nous présentons de nouvelles stratégies de recherches adaptées aux spécificités des nombres flottants pour résoudre plus efficacement les problèmes de vérification. Ces stratégies sont en cours d'évaluation sur un ensemble de benchmarks. Pour cela, nous avons porté le solveur FPCS [2] (Floating Point Constraint Solver) [13] développé par Claude Michel dans Objective-CP. Objective-CP est un outil d'optimisation introduit dans [17]. Ce portage nous permet de bénéficier de l'ensemble des facilités (e.g continuations, contrôleurs) offertes par Objective-CP pour implanter ces stratégies de recherche.

L'article est organisé comme suit : la section suivante présente quelques notations et définitions nécessaires à la compréhension de ce document. La section 3 explicite plusieurs propriétés basées sur les domaines des variables du système, puis des propriétés définies sur les contraintes. Ces propriétés servent de base aux nouvelles stratégies de recherches explicitées dans la section 4. La section 5 présente les choix liés à l'implantation des stratégies de recherche et enfin la section 6 discute des travaux en cours et des perspectives.

2. `http://www.i3s.unice.fr/~cpjm/misc/fpcs.html`

2 Notations et définitions

2.1 Les nombres à virgule flottante

Les nombres à virgule flottante ont été introduits pour approximer les nombres réels dans un ordinateur. La norme IEEE 754 [8] spécifie la représentation, ainsi qu'un ensemble de propriétés arithmétiques spécifiques aux flottants. Les deux principaux formats de représentations des flottants introduits dans IEEE 754 sont les formats *simple* et *double*. Selon le format utilisé, le nombre de bits de représentation est de 32 bits pour les flottants à simple précision, et de 64 bits pour les flottants à double précision. Un nombre à virgule flottante est composé de trois parties : le **signe** s (0 ou 1), la **mantisse** m et **l'exposant** e [7]. Un flottant normalisé est représenté par :

$$(-1)^s 1.m \times 2^e$$

Pour représenter les nombres flottants ayant une valeur absolue très petite, la norme IEEE 754 introduit la notion de nombre dénormalisé. Un flottant dénormalisé est représenté par :

$$(-1)^s 0.m \times 2^0$$

La **mantisse** m est représentée par 23 bits pour les flottants à simple précision, et par 52 bits pour le format double précision. Dans la suite de l'article, la **taille** de mantisse sera notée p.
L'exposant e est représenté par 8 bits pour le format simple (11 bits pour le format double).

2.2 Absorption

L'absorption est un phénomène qui apparait lors de l'addition de deux flottants d'ordres de grandeur très différents. Le résultat de l'addition est alors le flottant le plus grand.

Par exemple, en C, sous unix avec le format simple précision et un arrondi au plus près :

$$10^8 + 1.0 = 10^8$$

Dans cet exemple, la valeur 1.0 est absorbée par la valeur 10^8.

2.3 Cancellation

La cancellation correspond à une élimination des chiffres significatifs de poids les plus élevés. Elle apparait lors de la soustraction de deux flottants très proches résultant d'un calcul. Le phénomène est d'autant plus important lors d'une accumulation de calculs avec des erreurs arrondis. Cette accumulation d'erreurs de calculs est mise en évidence par la soustraction qui est exacte lorsque les opérandes sont proches [16].

Par exemple, en C, sous unix avec le format simple précision et un arrondi au plus près :

$$((1.0 - 10^{-8}) - 1.0) * 10^8 = 0$$

Dans cet exemple, la soustraction entre les valeurs correspondant au résultat de $(1.0 - 10^{-8})$ et 1.0 entraine une perte des chiffres significatifs. Or, le résultat de la soustraction est utilisé dans une multiplication pour illustrer le phénomène de cancellation intervenu lors de la soustraction. Le résultat, ici, devrait être -1 alors qu'il est de 0.

2.4 Notations

Dans la suite de cet article, x, y, z dénotent des variables flottantes, et D_x, D_y, D_z leurs domaines associés. De la même manière, $[\underline{x}, \overline{x}]$ représente tous les flottants compris entre les valeurs $\underline{x}$ et $\overline{x} \in F$, où F représente l'ensemble des flottants. Pour un nombre flottant f, f^+ et f^- dénotent respectivement le successeur et le prédécesseur de f. Enfin $|D_x|$ dénote la cardinalité du domaine de la variable x.

3 Propriétés sur les domaines des variables et des contraintes

Cette section définit plusieurs propriétés sur les domaines des variables et sur les contraintes. Ces propriétés sont utilisées pour construire de nouvelles stratégies de recherche. Les propriétés sur les domaines des variables telles que la cardinalité ou la densité capturent la structure des domaines des variables flottantes du problème. Les propriétés sur les contraintes prennent en compte les problèmes arithmétiques des flottants comme l'absorption ou la cancellation. Elles cherchent aussi à saisir la structure du problème à travers des propriétés telles que la dérivée.

3.1 Propriétés basées sur les domaines des variables

Les propriétés explicitées dans cette section servent de base aux nouvelles stratégies de recherches proposées. Ces propriétés se concentrent sur le domaine des variables du système de contraintes. Chaque propriété contient une description, une définition, et ses intérêts.

3.1.1 La taille

Description $width(D_x)$ dénote la taille du domaine de la variable x.

Définition

$$width(D_x) = \overline{x} - \underline{x} \quad (1)$$

Intérêts La taille du domaine correspond à la distance entre les deux bornes. C'est un critère plutôt historique. En effet, sur les domaines finis, de nombreuses stratégies sont basées sur ce critère, notamment une des plus populaires, minDom [12]. La sélection des variables avec le plus petit domaine dans les stratégies sur les entiers a pour but de favoriser les variables qui contraignent le plus le problème. Sur les flottants, ce critère est plus problématique à cause de la répartition non uniforme des flottants. Pour des domaines de taille différente, le domaine le plus petit n'est pas forcément le domaine avec le moins de flottants. Par exemple, si x et y sont deux variables ayant pour domaine respectif, $D_x = [1, 2]$ et $D_y = [10, 12]$, le plus petit domaine en termes de taille étant D_x. Or c'est D_y qui contient le moins de flottants ($|D_x| = 8388608$ et $|D_y| = 2097152$ alors que $width(D_x) = 1$ et $width(D_y) = 2$). Le critère de taille et celui de la cardinalité sont identiques sur domaines répartis de manière uniforme comme sur les domaines finis alors que dans le cas des flottants ces deux critères divergent.

3.1.2 La cardinalité

Description $|D_x|$ dénote la cardinalité du domaine de la variable x.

Définition Soit $D_x = [\underline{x}, \overline{x}]$ et $\underline{x} > 0$

$$|D_x| = 2^p * (e_{\overline{x}} - e_{\underline{x}}) + m_{\overline{x}} - m_{\underline{x}} + 1 \quad (2)$$

où $m_{\underline{x}}$ et $m_{\overline{x}}$ sont les mantisses de $\underline{x}$ et $\overline{x}$, $e_{\underline{x}}$ et $e_{\overline{x}}$ sont les exposants de $\underline{x}$ et $\overline{x}$

Cette formule est aisément extensible par symétrie aux autres cas.

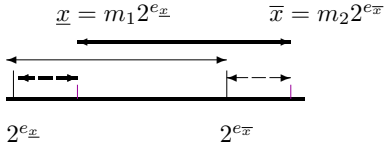


FIGURE 2 – Illustration de la formule

L'objectif est de calculer le nombre de flottants contenus dans l'intervalle D_x. Cet intervalle est représenté par la flèche en gras sur la figure 2. L'idée est donc de calculer le nombre de flottants contenus dans l'intervalle $[2^{e_{\underline{x}}}, 2^{e_{\overline{x}}}]$ (par la formule $(2^p * (e_{\overline{x}} - e_{\underline{x}}))$)

représenté par la flèche continue. Puis, de retirer le nombre de flottants compris entre $[2^{e_{\underline{x}}}, \underline{x}]$ (par la formule $-m_{\underline{x}}$) représenté par la flèche en gras et discontinue. Enfin il faut ajouter le nombre de flottants contenus dans l'intervalle $[\overline{x}, 2^{e_{\overline{x}}}]$ (par la formule $+m_{\overline{x}}$) représenté par la flèche droite discontinue. Il est nécessaire de comptabiliser un flottant supplémentaire $(+1)$ car la soustraction de la partie droite a supprimé le flottant $\underline{x}$.

Intérêts La taille et la cardinalité des domaines sont deux notions différentes sur les flottants. L'utilité d'utiliser la cardinalité pour guider la recherche est double. Tout d'abord, elle permet d'identifier les variables dont les domaines compte le moins de flottants dans le problème et donc les variables qui contraignent le plus le problème. Elle permet aussi de savoir quelle variable a le domaine le plus grand en nombre de flottants, ce qui permet de sélectionner des variables qui ont potentiellement beaucoup de valeurs appartenant aux solutions.

3.1.3 La densité

Description Cette propriété caractérise la proximité entre les flottants au sein du domaine d'une variable. On notera $\rho(D_x)$, l'application de la propriété au domaine de la variable x.

Définition

$$\rho(D_x) = \frac{|D_x|}{width(D_x)} \tag{3}$$

Intérêts La densité d'une variable peut permettre d'identifier les variables dont le domaine a des valeurs disparates (densité faible). La sélection de variables dont le domaine a une faible densité permet d'atteindre des valeurs sensiblement différentes et donc d'obtenir des comportements potentiellement différents au sein du système de contraintes. Dans le cas où les variables ont des domaines à forte densité, le nombre de valeurs du domaine potentiellement solution est plus conséquent. La densité d'un domaine augmente à proximité de 0.

3.1.4 La magnitude

Description Cette propriété permet d'identifier si le domaine est constitué uniquement de grands flottants ou de petit flottants. Par exemple, pour la magnitude du domaine $[0, 1]$, une valeur proche de 0 est attendue. La magnitude du domaine $[10^{36}, 10^{37}]$ quant à elle doit avoir une valeur proche de 1. Soit $mag(D_x)$, l'application de la propriété magnitude au domaine de la variable x.

Définition

$$mag(D_x) = \frac{e_{\underline{x}} + e_{\overline{x}}}{2 * e_{max}} \tag{4}$$

où e_{max} est l'exposant le plus grand.

Intérêts Cette propriété a un double objectif. Dans un premier temps, cette propriété peut être un moyen de sélectionner des variables qui sont potentiellement impliquées dans une absorption en combinant la sélection de variables à forte magnitude avec la sélection de variables à faible magnitude. Dans ce cas, cette propriété est simple à implanter, mais moins précise que la propriété d'absorption définie dans la section 3.2.1. Dans un second temps, à l'aide de cette propriété, les variables ayant des domaines avec des valeurs extrêmes peuvent être testées. Les valeurs extrêmes sont des valeurs particulières qui sont susceptibles d'entrainer des comportements non désirés.

3.2 Propriétés basées sur les contraintes

En plus des propriétés basées sur les domaines des variables, nous proposons des propriétés qui prennent en compte la structure des contraintes. Comme dans la section précédente, chacune des propriétés contient une description, une définition, et les intérêts. Ces propriétés sont de natures différentes, certaines sont des prédicats, d'autres retournent des valeurs.

3.2.1 Absorption

Description retourne vrai si la contrainte peut mener à une absorption, faux sinon. Cette propriété ne s'applique qu'aux contraintes d'additions. On notera cette propriété **canLeadToAnAbsorption**.

Définition Soit la contrainte $x + y = z$ avec y absorbée et un arrondi au plus près la condition :

$$canLeadToAnAbsorption = D_y \subseteq [\frac{x_m - x_m^+}{2}, \frac{x_m + x_m^+}{2}] \tag{5}$$

où $x_m = max(abs(\underline{x}), abs(\overline{x}))$ avec $abs(\underline{x})$ la valeur absolue de $\underline{x}$

Intérêts L'idée de cette propriété est de réduire la sélection des variables à celles impliquées dans des absorptions. Les problèmes d'absorptions sont récurrents lors de calcul sur les flottants. Orienter la recherche en prenant en compte ce phénomène ne peut qu'aider la recherche de contre-exemples lorsque l'absorption est mise en cause dans la correction du programme.

3.2.2 Cancellation

Description Cette propriété ne s'applique qu'aux contraintes de soustractions. Notons cette propriété **canLeadToACancellation**. La formule 6 est tirée de [2].

Définition Soit z = x - y

$$canLeadToACancellation = max(e^x, e^y) - e^z \quad (6)$$

Intérêts Cette propriété identifie les contraintes qui peuvent être impliquées dans une cancellation. Elle capture le taux de cancellation d'une contrainte. Plus le taux de cancellation est fort, plus la perte de chiffre significatif est forte. Si le taux est égal à zéro, il n'y a pas de cancellation. Tout comme les problèmes d'absorption, les problèmes de cancellation sont récurrents dans les programmes de vérification sur les flottants. Guider la recherche en tenant compte de ces phénomènes doit permettre de trouver plus efficacement un contre-exemple pour ces problèmes.

3.2.3 La dérivée

Description calcule la dérivée de la contrainte. On notera cette propriété **derivative**.

Définition

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \quad (7)$$

avec h est un flottant proche de 0.

Intérêts Dans une contrainte mono-variée, lorsque la dérivée est proche de 0, les valeurs de f(x) forment un palier. Le nombre de valeurs de x pour lesquelles la valeur de $f(x)$ est constante est donc potentiellement plus important. Dans ce cas, il y a plus de chance de trouver une solution. A contrario, lorsque la dérivée tend vers l'infini, il y a moins de valeurs solution. Un raisonnement identique peut-être fait sur les contraintes multivariées. L'objectif des stratégies qui utilisent la dérivée est de permettre de sélectionner une zone ayant potentiellement beaucoup de solutions ou au contraire très peu.

3.2.4 Le degré

Description La propriété **degré** est identique à la propriété sur les entiers. Elle se concentre sur le nombre de contraintes dans lesquels apparait la variable.

Définition

$$\sum_{i=0}^{|C|} implied(x, C_i) \quad (8)$$

où
$|C|$ est le nombre de contraintes du système
$implied(x, C_i)$ est un prédicat qui vaut 1 si la variable x est impliquée dans la contrainte C_i

Intérêts Cette propriété identifie les variables qui ont un impact fort sur la résolution du problème. Sur les domaines finis, de nombreuses stratégies utilisent cette propriété telle que weighted degree [3].

3.2.5 Les occurrences

Description Cette propriété compte le nombre maximal d'occurrences d'une variable dans les contraintes du problème.

Définition Soit $S = \{\forall\ C_i \in C,\ count(x, C_i)\}$

$$max(S) \quad (9)$$

avec
$count(x, C_i)$ est une fonction qui compte le nombre d'occurrences de x dans C_i
max est une fonction qui retourne le maximum.

Intérêts Les occurrences multiples d'une variable sont une problématique récurrente dans la vérification de programmes basés sur la programmation par contraintes. De nombreux travaux ont été faits pour gérer ce critère surtout en termes de consistance [11, 1]. L'idée ici est donc d'essayer de capturer l'importance d'une variable au sein d'une contrainte. En plus de permettre d'adapter le type de consistance au nombre d'occurrences présente au sein d'une contrainte, ce critère permet d'identifier les variables qui ont un impact fort sur une contrainte spécifique.

4 Nouvelles stratégies de recherche

Dans cette section, nous proposons un ensemble de stratégies de recherche basées sur les propriétés des sections 3.1 et 3.2. Dans un premier temps, nous définissons des stratégies simples qui se concentrent sur une seule propriété. Puis dans un second temps, nous introduisons des stratégies qui tentent d'exploiter plusieurs propriétés en même temps.

4.1 Stratégies mono-propriété

Les stratégies mono-propriété proposées suivent toutes le même schéma. Pour chaque propriété basée sur les variables, deux stratégies sont définies : une qui sélectionne la variable qui minimise la propriété et l'autre qui sélectionne la variable qui la maximise. Par exemple, pour la propriété cardinalité, deux stratégies sont obtenues, la première sélectionne la variable qui minimise la cardinalité, la seconde sélectionne la variable qui maximise la cardinalité lors de la sélection de la variable. Pour la propriété magnitude, une stratégie supplémentaire est définie qui alterne entre la sélection de variable qui minimise la magnitude et celle qui la maximise. Cette stratégie a pour objectif d'essayer de déclencher artificiellement les phénomènes d'absorption.

Pour les propriétés basées sur les contraintes, les propriétés *degree*, *occurrences* et *derivative*, sont utilisées pour définir deux stratégies. Ces deux là sont :

1. Pour les propriétés *degree* et *occurences*, les stratégies sont similaires à celles pour les variables. Une minimise le critère, alors que la seconde le maximise.

2. Pour la propriété *derivative*, la première sélectionne la variable qui est impliquée dans le plus de contraintes dont la dérivée est proche de 0. Cette sélection cherche les variables ayant potentiellement beaucoup de valeurs solutions. La seconde sélectionne d'abord celle qui est le plus impliquée dans les contraintes dont la dérivée tend vers l'infini.

Changer de stratégie durant la recherche peut-être nécessaire. Certaines propriétés sur les contraintes sont statiques (e.g degree et occurences), d'autres sont dynamiques (e.g la dérivée) ou passent de faux à vrai une seule fois lors de la descente dans l'arbre de recherche (e.g canLeadToAnAbsorption). Cette remarque est également vraie pour les stratégies de la section 4.2.2. Lorsqu'une propriété n'a plus d'intérêt, il est important d'en changer pour en choisir une qui améliore la recherche.

4.2 Stratégies multi-propriétés

Dans cette section, différentes propriétés sont combinées. Une première manière de combiner les propriétés définies dans les sections 3.1 et 3.2 est de les agréger avec une formule. Une seconde manière de les combiner est de les appliquer les unes à la suite des autres.

4.2.1 Stratégies avec score

Cette section propose la mise en relation de plusieurs propriétés en utilisant une formule. Cette proposition est inspirée de la stratégie de recherches IBS [15] (Impact Based Search). IBS est une stratégie qui mesure l'impact de l'affectation d'une variable en définissant un score. Ce score correspond à la réduction de l'espace de recherche suite à la propagation de l'affectation de la variable. Ce score peut-être diminué ou augmenté selon l'efficacité du choix précédemment effectué lors de la recherche. À chaque étape, la variable ayant le meilleur score est choisie. Les stratégies présentées dans cette section reprennent cette idée de privilégier en premier les variables qui satisfont au mieux différents scores. Les grandes lignes de la sélection de variables sur le principe d'un score sont les suivantes :

1. Calculer le score de chaque variable.
2. Sélectionner la variable qui maximise le score.

Le premier score S_1 présenté essaye de lier la propriété de taille à celle de cardinalité. Le choix de ces deux propriétés est du à leur proximité. En effet, l'une définit la distance les bornes du domaine et l'autre le nombre de flottants présents dans ce même domaine.

$$S_1(x) = \omega_1|D_x| + \omega_2\rho(D_x)$$

Les ω_i représentent des pondérations. Ces pondérations seront à définir lors des expérimentations.

Le second score S_2 réunit les mêmes propriétés que S_1 en ajoutant le degré de la variable.

$$S_2(x) = -\frac{S_1(x)}{degree(x)}$$

Ce choix tente de prendre en compte les variables qui contraignent le plus le problème pour avoir un plus grand impact sur la résolution.

4.2.2 Autres stratégies multi-propriétés

Dans cette section, nous proposons des stratégies qui allient les propriétés basées sur les contraintes, sur les domaines des variables, et les scores. Combiner ces stratégies de recherche permet d'être plus fin au niveau de la recherche en prenant en compte plusieurs critères. Par exemple, utiliser une stratégie de recherche basée sur une propriété statique ne prend en compte que la structure initiale du problème. Or, la descente dans l'arbre de recherche fait qu'on est confronté à un sous-problème du problème initial où la propriété peut ne plus être significative. Pour ce faire, les propriétés définies sur les contraintes sont utilisées pour réduire le choix des variables à celle qui ont une structure particulière (e.g qui peuvent mener à une absorption). Une fois cette réduction effectuée, les propriétés établies

sur les variables sont utilisées pour sélectionner celle qui nous intéresse le plus. L'utilisation d'un score peut aussi remplacer la propriété sur les variables.

Le premier groupe de stratégies par propriétés sur les contraintes que nous proposons. Le premier utilise la propriété **canLeadToAnAbsorption** pour réduire le choix des variables à celle impliquée dans une absorption. À la suite de l'application de cette propriété, une des stratégies mono-propriété est utilisée. L'idée de ce groupe de stratégies est le suivant :

1. Extraire les variables qui peuvent mener à une absorption (en utilisant le prédicat canLeadToAnAbsorption).
2. Appliquer la propriété choisie aux variables extraites.
3. Sélectionner la variable qui maximise ou minimise la propriété.

Le second groupe de stratégies proposé, repose sur la propriété **canLeadToACancellation** pour restreindre le choix des variables à celles qui sont impliquées dans une cancellation. La propriété canLeadToACancellation calcule un taux de cancellation. La restriction peut donc être modulée en choisissant le taux minimal d'acceptation d'une variable (e.g toutes les variables impliquées dans une cancellation avec un taux supérieur à 0.5). Suite à l'application de cette propriété, une des stratégies définies à la section 4.1 est appliquée.

Enfin pour les propriétés **degree**, **occurrences** et **dérivative** deux choix sont à évaluer. Le premier consiste à minimiser la propriété choisie, le second à la maximiser. Une fois le choix appliqué, une des stratégies monopropriété est mise en œuvre. On peut également envisager de composer d'autres propriétés sur les contraintes par exemple *canLeadToAnAbsorption* ou *canLeadToACancellation* à la suite de propriétés choisies, pour ensuite finir par une propriété sur les variables.

Ces stratégies peuvent également être combinées en les appliquant à des sous-domaines. Par exemple, une fois une variable sélectionnée, il est possible de diviser le domaine de la variable en plusieurs sous-domaines pour appliquer différentes stratégies sur chaque sous-domaines.

5 Implantation

Les stratégies présentées sont en cours d'évaluation sur un ensemble de benchmarks. Pour ce faire, le solveur FPCS (Floating Point Constraint Solver) [13] développé par un des auteurs a été porté dans Objective-CP. Objective-CP est un outil d'optimisation introduit dans [17]. Ce portage a été effectué pour apporter une flexibilité dans le développement des stratégies de recherche que nous proposons. Dans cette section nous discutons des choix d'implantation effectués.

5.1 Choix du solveur

Pour explorer de nouvelles stratégies de recherche sur les problèmes de vérification, nous avons besoin d'un solveur sur les flottants et d'un solveur avec la possibilité de définir des stratégies flexibles. Un solveur sur les flottants FPCS a été développé au sein de notre équipe de recherche. Ce solveur est basé sur un filtrage fort. Cependant, l'écriture de stratégie de recherche dans FPCS est une tâche longue et fastidieuse. Objective-CP quant à lui est une boite à outils d'optimisation sur les domaines finis, avec de nombreuses facilités pour le développement de stratégies de recherche flexible. Pour avoir toute l'efficacité de FPCS et la flexibilité des stratégies de recherche d'Objective-CP nous avons décidés d'incorporer FPCS dans Objective-CP. Ces deux outils sont complémentaires. Dans les deux parties qui suivent, nous présentons d'abord ces deux solveurs, leurs avantages et limites.

5.1.1 FPCS

Présentation FPCS est un solveur de contraintes sur les flottants conçu et développé par C.Michel. Ce solveur est écrit en C++, il a été développé dans la perspective de la vérification de programme.

Avantages FPCS a de nombreux avantages. C'est un solveur complet et efficace supportant les filtrages fort tel que la KB-consistance [11], les différents modes d'arrondi, et un catalogue de contraintes issu des problèmes de vérification de programmes avec flottants. De plus, l'ajout des contraintes au modèle est simple (voir la figure 3).

```
model.add(a = 2 * b + 3 * c);
```

FIGURE 3 – Ajout de contrainte dans FPCS

Limites Le principal défaut de FPCS est qu'il n'a pas été conçu pour développer des stratégies de recherche complexes, mais a plutôt vocation à être utilisé avec des stratégies simples comme la bissection. De ce fait, le code des stratégies de recherche peut vite devenir long et fastidieux et l'arbre de recherche doit être traité en ajoutant les différents points de choix un à un. Par exemple, 80 lignes de codes sont nécessaires pour implanter la bissection. De plus, l'ajout d'un point de choix entraine de nombreuses copies non désirées.

```
1       choice->domain->setLBTo(&low);
2       choice->domain->setUBTo(&low);
3       push();
4
5       choice->domain->setLBTo(&up);
6       choice->domain->setUBTo(&up);
7       push();
```

FIGURE 4 – Définition d'une stratégie de recherche dans FPCS

Dans l'exemple de la figure 4, les deux points de choix successifs correspondant aux bornes de l'intervalle courant sont ajoutés à la stratégie. Ici, *choice* est une variable. La première ligne force la borne inférieure du domaine de la variable *choice* à la valeur low. La seconde force la borne supérieure du domaine de la variable *choice* à la valeur low également. À la troisième ligne le choix de la réduction du domaine de la variable *choice* à $[low, low]$ est ajouté à la pile des choix de la recherche. De la cinquième ligne à la septième, le domaine de la variable *choice* est réduit à $[up, up]$ et le point de choix est ajouté.

5.1.2 Objective-CP

Présentation Objective-CP est un outil d'optimisation conçu par P. VanHentenryck et L.Michel [17]. Cet outil est développé en Objective-C et contient plusieurs solveurs, dont un solveur CP (Constraint Programming), un solveur LP (Linear Programming), et un solveur MIP (Mixed-Integer Programming).

Avantages Objective-CP est un outil qui sépare la partie modélisation de la partie résolution (solveur) et stratégie de recherche. En effet, lors de la modélisation du problème les variables et les contraintes définies sont abstraites. Le passage de la modélisation à la résolution s'effectue par la définition d'un programme (CPProgram ou LPProgram ou MIPProgram). Ce programme correspond au solveur (cf. figure 5) qui sera appelé pour concrétiser les variables et contraintes abstraites en variables et contraintes concrètes (interne à chaque solveur). Dans la figure 5, les deux premières lignes définissent un programme CP. Ce programme correspond au modèle concrétisé par un solveur CP. Aux deux dernières lignes, un processus identique est effectué avec un programme LP. La partie recherche est basée sur les concepts de continuations et de contrôleurs qui permettent l'écriture de stratégies flexibles, faciles à prendre en main et concises en termes de lignes de code. Par exemple, la figure 6 montre comment implanter la bissection en 10 lignes. Enfin, cet outil incorpore également une partie qui permet de paralléliser en quelques lignes les solveurs.

```
id<CPProgram> p;                                    1
p = [ORFactory createCPProgram:model];              2
                                                    3
id<LPProgram> p2;                                   4
p2 = [ORFactory createLPProgram:model];             5
```

FIGURE 5 – Appel au solveur CP

```
while (![xi bound]) {                               1
  ORFloat theMax = xi.max;                          2
  ORFloat theMin = xi.min;                          3
  ORFloat mid = (theMin + theMax)/2.0f;             4
  if(mid == theMax)                                 5
        mid = theMin;                               6
  [_search try: ^{                                  7
    [self floatGthenImpl:xi with:mid];              8
    }                                               9
   alt: ^{                                          10
    [self floatLEqualImpl:xi with:mid];             11
    }                                               12
   ];                                               13
}                                                   14
```

FIGURE 6 – Exemple d'une stratégie de recherche avec bissection en Objective-CP

Dans la figure 6, aux seconde et troisième lignes les bornes de la variable xi sont extraites. La quatrième ligne correspond au calcul du milieu de l'intervalle *mid*. Le test à la cinquième ligne permet de gérer le cas où il n'y a que 3 flottants. La septième ligne correspond au branchement. Le *try* représente la branche où $xi > mid$. Le *alt* correspond à la branche $xi \leq mid$. Ce processus est répété tant que le domaine de x_i n'est pas dégénéré.

Limites Le solveur CP implanté dans Objective-CP ne supporte pas les flottants donc tout un travail d'implantation est nécessaire pour l'ajout de la partie flottante. Du fait que Objective-CP est basé sur Objective-C l'ajout des contraintes est moins élégant que dans FPCS (figure 7), mais dans le cadre de la vérification de programme les modèles sont généralement créés de manière automatique.

```
[model add:[a eq:[[b mul:@(2)]
                 plus:[c mul:@(3)]]]];
```

FIGURE 7 – Ajout de la contrainte $a = 2b + 3c$ en Objective-CP

6 Discussions et perspectives

Cet article propose plusieurs stratégies de recherche dédiées aux systèmes de contraintes sur les flottants, et plus particulièrement aux problèmes de vérification de programme avec du calcul sur les flottants. Les différentes stratégies proposées sont basées sur un ensemble de propriétés basées sur deux critères : les domaines des variables et les contraintes. Les premières cherchent à capturer la structure spécifique des flottants telles que leurs nombres ou leurs densités. Les secondes tendent à prendre en compte les problèmes liés à l'arithmétique propre aux flottants et à la structure même des problèmes. Un travail d'expérimentation pour évaluer ces stratégies sur un ensemble d'exemple de problèmes liés à la vérification de programme est actuellement en cours.

Références

[1] Mohammed Said Belaid, Claude Michel, and Michel Rueher. Boosting local consistency algorithms over floating-point numbers. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, pages 127–140, 2012.

[2] Florian Benz, Andreas Hildebrandt, and Sebastian Hack. A dynamic program analysis to find floating-point accuracy problems. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12, Beijing, China - June 11 - 16, 2012*, pages 453–462, 2012.

[3] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. *ECAI'04*, pages 146–150, 2004.

[4] Hélène Collavizza, Michel Rueher, and Pascal Van Hentenryck. Cpbpv : A constraint-programming framework for bounded program verification. *Constraints*, 15(2) :238–264, 2010.

[5] Hélène Collavizza, Nguyen Le Vinh, Michel Rueher, Samuel Devulder, and Thierry Gueguen. A dynamic constraint-based BMC strategy for generating counterexamples. *26th ACM Symposium On Applied Computing*, 2011.

[6] Steven Gay, Renaud Hartert, Christophe Lecoutre, and Pierre Schaus. Conflict ordering search for scheduling problems. In *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, pages 140–148, 2015.

[7] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1) :5–48, 1991.

[8] IEEE. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard*, 754, 2008.

[9] Narendra Jussien and Olivier Lhomme. Dynamic domain splitting for numeric csps. *ECAI*, pages 224–228, 1998.

[10] R. Baker Kearfott. Some tests of generalized bisection. *ACM Trans. Math. Softw.*, 13(3) :197–220, September 1987.

[11] Olivier Lhomme. Consistency techniques for numeric csps. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'93, pages 232–238, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[12] Jeff T. Linderoth and Martin W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2) :173–187, 1999.

[13] C. Michel, M. Rueher, and Y. Lebbah. Solving constraints over floating-point numbers. In *Principles and Practice of Constraint Programming — CP 2001 : 7th International Conference, CP 2001 Paphos, Cyprus, November 26 – December 1, 2001 Proceedings*, pages 524–538, Berlin, Heidelberg, 2001.

[14] Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. In Nicolas Beldiceanu, Narendra Jussien, and Éric Pinson, editors, *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems : 9th International Conference, CPAIOR 2012, Nantes, France, May 28 – June1, 2012. Proceedings*, pages 228–243, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[15] Philippe Refalo. Impact-based search strategies for constraint programming. *CP 2004*, 3258 :557–571, 2004.

[16] P.H. Sterbenz. *Floating-point computation.* Prentice-Hall series in automatic computation. Prentice-Hall, 1973.

[17] Pascal Van Hentenryck and Laurent Michel. The objective-cp optimization system. In *Principles and Practice of Constraint Programming : 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, pages 8–29, Berlin, Heidelberg, 2013.

Une contrainte de circuit adaptée aux tournées multiples

Nicolas Briot[1] * **Philippe Vismara**[2] **Christian Bessiere**[3]

[1] Université Montpellier - LIRMM, Montpellier, France
[2] Montpellier SupAgro - LIRMM MISTEA, Montpellier, France
[3] CNRS - LIRMM, Montpellier, France
{briot,vismara,bessiere}@lirmm.fr

Résumé

Il existe de nombreuses applications réelles contenant un problème de tournées de véhicules. La programmation par contraintes permet d'aborder ces problèmes de façon efficace. Des contraintes de circuits ont été définies pour traiter du problème de voyageur de commerce (TSP) ou de tournées de véhicules (VRP). Ces contraintes sont basées sur la recherche d'un circuit hamiltonien dans un graphe. Dans cet article, nous nous intéressons au problème plus général de tournées multiples dans lequel on cherche à couvrir une partie du graphe par un ensemble de circuits de coût minimal. Nous proposons une nouvelle contrainte globale basée sur la recherche de circuits élémentaires disjoints dans un graphe. Contrairement aux contraintes existantes, on ne cherche pas un circuit hamiltonien mais un ensemble de circuits de Steiner. Après avoir défini cette contrainte, nous montrons que le filtrage au bornes est NP-difficile. Nous proposons donc une méthode de filtrage incomplète basée sur la recherche d'une borne inférieure d'un circuit de Steiner.

De nombreux problèmes réels peuvent être ramenés à des problèmes de tournées. Les problèmes de tournées forment une grande famille dont les deux représentants les plus connus sont le problème du voyageur de commerce (TSP) et le problème de tournées de véhicules (VRP). Le premier consiste à trouver un circuit hamiltonien de coût minimum dans un graphe et le second est sa généralisation à plusieurs circuits avec un sommet dépôt commun. Ces problèmes se déclinent souvent avec d'autres contraintes comme par exemple, des fenêtres de temps ou des précédences sur les sommets du graphe ou encore des contraintes de taille sur les circuits. Tous ces problèmes sont NP-difficiles.

Les problèmes de base (sans contraintes particulières) ont été étudiés de manière intensive en recherche opérationnelle. Ainsi, les méthodes de résolution exactes les plus efficaces actuellement sont basées soit sur la programmation dynamique soit sur la programmation linéaire en nombres entier (ILP).

L'ajout de contraintes spécifiques au problème peut rendre l'approche ILP insuffisante du point de vue de l'expressivité ou de la résolution. La programmation par contraintes (CP) offre un cadre général plus à même de capturer et exploiter ces contraintes spécifiques pour résoudre le problème. Pour les problèmes de tournées, des contraintes de recherche de circuits dans un graphe ont été créées, mais elles ne suffisent pas toujours. Comme pour le problème du circuit de Steiner où l'ensemble de sommets à couvrir n'est pas fixe.

Après un rapide survol des principes de la programmation par contraintes, cet article énumère les contraintes de circuits ainsi que les principaux algorithmes de filtrage qui existent à ce jour. Nous étudions leurs limites et proposons une nouvelle contrainte capable d'y répondre. Nous montrons la complexité de la nouvelle contrainte et nous donnons les premiers algorithmes de filtrage.

1 Préliminaires

En CP, chaque variable $x \in X$ est associée à un domaine $dom(x)$ qui correspond à l'ensemble des valeurs que peut prendre la variable. Les domaines sont restreints par un ensemble de contraintes C. Une contrainte est une relation qui porte sur un ensemble de variables et qui définit l'ensemble des valeurs que peuvent prendre simultanément les variables. Une so-

*Papier doctorant : Nicolas Briot[1] est auteur principal.

lution au problème est donnée quand tous les domaines sont réduits à un singleton (les variables sont alors instanciées) et quand toutes les contraintes sont satisfaites. Une contrainte est satisfaite si l'ensemble des valeurs prises par les variables satisfait la relation.

En CP, des contraintes dites globales permettent de capturer une partie de la structure combinatoire d'un problème. Chacune de ces contraintes est associée à un algorithme de filtrage qui va détecter des valeurs impossibles dans le domaine des variables. Il est important que le filtrage soit rapide et efficace. Pour cela, plusieurs niveaux de filtrage ont été définis. La cohérence d'arc généralisée (GAC) est un haut niveau de filtrage qui traite les contraintes indépendamment.

Définition. *Étant donné une contrainte $c \in C$ qui porte sur les variables $(x_1, ..., x_n)$ de domaines respectifs $dom(x_1), ..., dom(x_n)$, la contrainte c est cohérente d'arc généralisée (GAC) si et seulement si $\forall x_k \ k \in 1..n$ et $\forall v \in dom(x_k)$, il existe un tuple $(v_1, ..., v, ..., v_n)$ avec $v_i \in dom(x_i)$ qui satisfait la contrainte.*

Appliquer GAC peut être exponentiel sur certaines contraintes globales. Moins forte, la cohérence aux bornes (BC) peut être une option plus abordable.

Définition. *Étant donné une contrainte $c \in C$ qui porte sur les variables $(x_1, ..., x_n)$, la contrainte c est cohérente aux bornes (BC) si et seulement si $\forall x_i \in c$: les tuples $(v'_1, ..., LB(v_i), ..., v'_n)$ et $(v_1, ..., UB(v_i), ..., v_n)$ satisfont la contrainte avec $LB(x_k)$ (resp. $UB(x_k)$) la valeur minimale (maximale) de $dom(x_k)$ et v_j, v'_j $(j \neq i)$ des nombres dans l'intervalle $[LB(v_j), ..., UB(v_j)]$.*

2 Les contraintes de circuit

Considérons un graphe orienté $G = (V, A, c)$, avec $V = \{1, ..., n\}$ l'ensemble des sommets et A l'ensemble des arcs valués par une fonction de coût c.

En CP, la recherche de circuits hamiltoniens dans G peut être modélisée en posant des contraintes globales sur un ensemble de variables décrivant le successeur de chaque sommet.

Pour tout sommet $i \in V$, notons $Next_i$, la variable qui représente le sommet successeur du sommet i. Le domaine $dom(Next_i)$ correspond à l'ensemble des sommets adjacents à i dans le graphe, c'est-à-dire $j \in dom(Next_i)$ si et seulement si $(i, j) \in A$.

Par définition, les variables successeurs $Next_i$ décrivent un sous-graphe orienté. Pour les graphes non-orientés, une arête peut être représentée par deux arcs entre les deux sommets. Dans ce cas, la recherche d'un cycle dans un graphe non-orienté correspond à la recherche d'un circuit dans le graphe orienté équivalent. Dans la suite, nous désignerons indifféremment par TSP le problème du voyageur de commerce orienté (symétrique) ou non-orienté (asymétrique).

La contrainte

$$\textsc{Circuit}(Next_1, ..., Next_n) \text{ [1]}$$

est satisfaite si et seulement si l'ensemble des arcs définis par $\{(i, Next_i) \in A\}$ forme un circuit hamiltonien dans G.

Établir la GAC pour cette contrainte est NP-difficile puisque cela revient à chercher des circuits hamiltoniens dans un graphe.

Des approches incomplètes ont donc été proposées pour le filtrage de la contrainte Circuit. Premièrement, la contrainte requiert que deux sommets n'aient pas le même successeur. Cette propriété correspond à la contrainte $\textsc{AllDifferent}(Next_1, ..., Next_n)$ qui possède un filtrage GAC efficace. Deuxièmement, l'élimination des sous-circuits a été proposée dans Pesant *et al.* [12] et Caseau *et al.* [3]. L'algorithme interdit les affectations du type $Next_i = j$ si l'arc (i, j) provoque un sous-circuit dans le sous-graphe induit par les arcs $\{(i, Next_i) \mid Next_i \text{ est instanciée}\}$. Il existe aussi un algorithme de filtrage moins utilisé basé sur la recherche de séparateur de graphes (voir [11]).

Une version de la contrainte prenant en compte le coût du circuit a été proposée en premier par Focacci *et al.* [7, 5, 6] puis par Benchimol *et al.* [2]. Cette contrainte a été développée pour répondre précisément au problème du TSP.

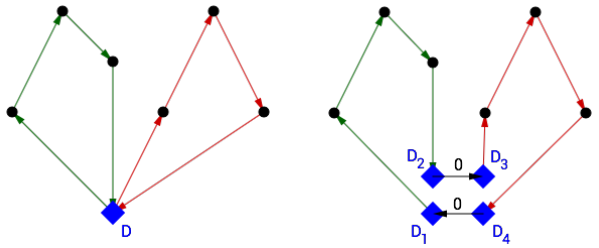
Soit Z une variable entière qui représente le coût du circuit hamiltonien. Cette fois, la contrainte

$$\textsc{WeightedCircuit}[G](Next_1, ..., Next_n, Z)$$

est satisfaite si et seulement si l'ensemble des arcs définis par $\{(i, Next_i) \in A\}$ forme un circuit hamiltonien dans G de coût $\leq Z$ où le coût du circuit est la somme des poids des arcs du circuit.

La première partie du filtrage de la contrainte WeightedCircuit est équivalente au filtrage de la contrainte Circuit.

Durant la recherche, des valeurs dans le domaine des variables successeurs seront retirées. Ces valeurs correspondent à des arcs du graphe de départ et, au fur et à mesure, le graphe potentiel se vide jusqu'à obtenir un circuit. Dès qu'un arc est retiré du graphe, la borne inférieure du coût du circuit $LB(Z)$ peut être mise à jour. Une partie du travail de l'algorithme de filtrage



(a) (b)

FIGURE 1 – Transformation d'un problème de VRP à deux tournées en un problème de TSP. Le nœud dépôt est dupliqué en autant de paires de sommets (départ/arrivée) que de tournées, afin de se ramener à un circuit hamiltonien.

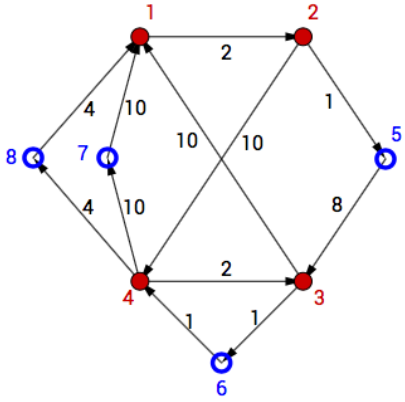


FIGURE 2 – Exemple du problème du circuit de Steiner. En rouge (plein) les sommets terminaux et en bleu (cercle) les sommets potentiels

est donc de trouver la meilleure borne possible pour $LB(Z)$. Cette borne peut être calculée grâce à deux relaxations du TSP : le problème de l'arborescence (ou arbre en version non-orienté) couvrante de coût minimum (MSA) et le problème d'affectation (AP). AP consiste à relâcher la contrainte d'élimination des sous-tours dans le TSP et revient donc à chercher un ensemble de circuits élémentaires couvrant le graphe et de coût minimum.

Benchimol *et al.* [2] ont proposé d'utiliser la relaxation 1-tree définie par Held et Karp [8, 9] qui donne une relaxation strictement meilleure que l'arbre couvrant de coût minimum. Ces deux relaxations (Held & Karp et AP) se calculent en temps polynomial. Ducomman *et al.* [4] ont montré que les bornes obtenues par ces filtrages étaient incomparables et que les filtrages pouvaient être utilisés indépendamment pour filtrer $LB(Z)$.

Disposer d'une bonne estimation de $LB(Z)$ permet également de filtrer les variables $Next_i$ en éliminant certains arcs incompatibles avec la valeur de $UB(Z)$ (voir Benchimol *et al.* [2] et Foccacci *et al.* [6]).

Les contraintes de circuit présentées ici sont limitées à la recherche d'un circuit hamiltonien. Elles s'appliquent donc assez bien à des problèmes comme le TSP, le TSP avec fenêtres de temps, etc. Modéliser un problème plus général comme le VRP avec une contrainte de circuit nécessite une modélisation plus complexe. La figure 1 illustre la transformation nécessaire pour passer d'un VRP à un TSP et pouvoir ainsi utiliser une contrainte de circuit. Cette modélisation impose d'avoir au moins un sommet fixé dans chaque circuit (ici, le sommet dépôt), ce qui n'est pas forcément le cas pour certains problèmes de couverture de sommets.

Les contraintes de circuit hamiltonien ne peuvent pas non plus être utilisées lorsque l'ensemble des sommets intervenant dans le circuit n'est pas totalement connu.

Considérons un graphe $G = (V, A, c)$ et soit $V^* \subseteq V$ un sous-ensemble de sommets dit obligatoires (ou terminaux). Le *problème du circuit de Steiner* dans G pour V^* consiste à trouver un circuit élémentaire dans G de coût minimum passant par tous les sommets de V^*. Le circuit de coût minimum peut aussi passer par des sommets extérieurs à V^*, ce sont les nœuds de Steiner. Le TSP est un cas particulier où $V^* = V$. Ce problème est fortement lié au problème plus connu de l'arbre de Steiner [10].

Exemple 1. *Soit $G = (V, A, c)$ le graphe de la figure 2 et $V^* = \{1, 2, 3, 4\}$ les sommets terminaux (obligatoires). La solution au circuit de Steiner est la séquence de sommets $(1, 2, 5, 3, 6, 4, 8, 1)$ de coût 21. Dans cette solution, le sommet 7 n'intervient pas dans le circuit optimal.*

3 La contrainte WeightedSubCircuits

Dans cette section, nous proposons une nouvelle contrainte qui répond aux limites des contraintes de circuit présentées précédemment. La nouvelle contrainte cherche au plus K circuits élémentaires disjoints de coût minimum couvrant une partie du graphe. Un circuit élémentaire est un circuit qui passe une et une seule fois sur chaque sommet du circuit. Dans la suite, nous admettons qu'une boucle sur un sommet est un circuit élémentaire (composé d'un arc).

Soit $G = (V, A, c)$ un graphe orienté avec $V = \{1, \ldots, n\}$ l'ensemble des sommets, $A = \{(i, j) \mid i, j \in$

$V\}$ l'ensemble des arcs et la fonction de coût $c : A \to \mathbb{N}$ associée à chaque arc de G.

Définition. *Étant données les variables :*

- Set_k $\forall k \in \{1, .., K\}$, *une variable ensembliste représentant les sommets de G qui sont dans le $k^{ème}$ circuit. On a $dom(Set_k) = 2^V$ (l'ensemble des parties de V).*
- Set_{dummy}, *la variable ensembliste représentant l'ensemble des sommets exclus de tous les circuits. $dom(Set_{dummy}) = 2^V$*
- $Next_i$ $\forall i \in V$, *représente le sommet successeur du sommet i. $dom(Next_i) \subseteq V$.*
- Z *et* $Cost_k$ $\forall k \in \{1, .., K\}$ *des variables entières qui représentent respectivement le coût total des circuits et le coût du circuit k.*

La contrainte WeightedSubCircuits, *notée*

$$\text{WSC}[G](\{Set_k\}_{k=1..K}, Set_{dummy} \{Next_i\}_{i=1..n}, \{Cost_k\}_{k=1..K}, Z) \quad (1)$$

est satisfaite si et seulement si les conditions suivantes sont respectées :

1. $\forall k, k' \in 1..K \cup dummy, \quad (k \neq k' \Rightarrow Set_k \cap Set_{k'} = \emptyset)$;
2. $Set_1 \cup ... \cup Set_K \cup Set_{dummy} = V$;
3. $\forall k \in 1..K$:
   - *si* $Set_k = \emptyset$ *alors* $Cost_k = 0$.
   - *sinon, $G[Set_k]$ le sous-graphe de G induit par l'ensemble de sommets Set_k et par l'ensemble d'arcs $\{(i, Next_i) \mid i \in Set_k\}$ est un circuit élémentaire de G de coût $\leq Cost_k$.*
4. $\forall i \in V$, $i \in Set_{dummy} \Leftrightarrow Next_i = i$;
5. $\sum_{k=1}^{K} Cost_k \leq Z$;

La contrainte WSC impose un ensemble d'au plus K circuits élémentaires disjoints dans G de coût total $\leq Z$. Les conditions 1 et 2 correspondent à une partition des sommets dont certaines parties peuvent être vides. La condition 3 autorise des circuits vides dont le coût associé vaut 0. Si l'ensemble possède au moins un sommet, alors le sous-graphe induit par les sommets de Set_k et les arcs $\{(i, Next_i) \mid i \in Set_k\}$ est un circuit élémentaire de G dont le coût ne dépasse pas $Cost_k$. S'il n'y a qu'un seul sommet i, le circuit est réduit à une boucle de coût $c(i, i)$.

Il faut noter qu'un sommet i est exclu des K circuits si et seulement si la valeur i est dans l'ensemble Set_{dummy} et la condition 4 associe la valeur i à $Next_i$. À l'inverse, si $Set_{dummy} = \emptyset$ alors tous les sommets sont impliqués dans les circuits. De même, si $i \notin D(Next_i)$ alors i doit être dans un circuit. Enfin, le coût total des circuits est borné par Z dans la condition 5.

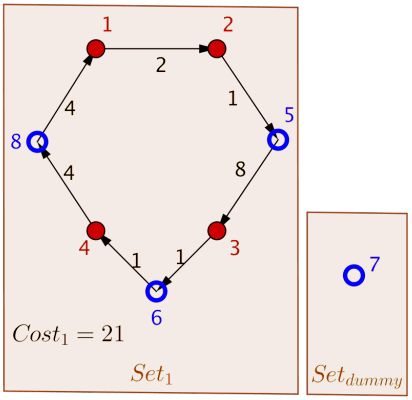


FIGURE 3 – La solution optimale au problème du circuit de Steiner sur le graphe de la figure 2. Les variables $Next_i$ sont représentées par les flèches et les ensembles par les rectangles.

Dans cet article, nous proposons une définition de la contrainte utilisant des variables ensemblistes pour décrire l'ensemble des sommets associés à chaque circuit. Une perspective interessante serait d'étudier d'autres représentations de ces ensembles, par exemple par des variables entières.

Exemple 2. *Reprenons le graphe de la figure 2. Pour $K = 1$ on modélise le problème du circuit de Steiner par le réseau de contraintes suivant :*

- $LB(Set_1) = \{1, 2, 3, 4\}$, $UB(Set_1) = V$;
- $LB(Set_{dummy}) = \emptyset$, $UB(Set_{dummy}) = V$;
- $Next_1 = 2, Next_2 \in \{4, 5\}, Next_3 \in \{1, 6\}, Next_4 \in \{7, 8\}, Next_5 \in \{3, 5\}, Next_6 \in \{4, 6\}, Next_7 \in \{1, 7\}$ *et* $Next_8 \in \{1, 8\}$;
- $Cost_1, Z \in \mathbb{N}$;
- $\text{WSC}[G](Set_1, Set_{dummy}, Next_1, .., Next_8, Cost_1, Z)$

sur lequel on cherche à minimiser Z.

Dans cet exemple, un seul circuit est recherché et $Cost_1$ peut être confondu avec Z. La solution optimale est représentée par la figure 3. Le circuit a un coût de 21. Le sommet 7 est exclu du circuit car il fait partie de l'ensemble Set_{dummy} et sa variable successeur $Next_7$ est instanciée à l'unique valeur 7 (non représentée sur la figure).

Théorème 1. *Établir BC et établir GAC pour la contrainte* WSC *est NP-difficile.*

Démonstration. Décider si un graphe $G = (V, A, c)$ admet un circuit hamiltonien de coût inférieur ou égal à une valeur p est NP-complet. Considérons la contrainte

$\text{WSC}[G](\{Set_1, Set_2\}, \{Next_i\}_{i=1..n}, \{Cost_1\}, Z)$, où $LB(Set_1) = V$, où chaque $Next_i$ a pour domaine l'ensemble des successeurs du sommet i dans A et où $Cost_1 \in \{0..p\}$, $Z \in \{0..p\}$. BC vide le domaine de Z si et seulement si le graphe G admet un circuit hamiltonien de coût inférieur ou égal à p. □

3.1 Décomposition

Pour implémenter le filtrage de la contrainte WeightedSubCircuits, une première approche consiste à remplacer la contrainte par un ensemble de contraintes équivalentes en termes d'expressivité. Cela permet de bénéficier directement des filtrages associés à ces contraintes. Nous verrons par la suite que cet ensemble de contraintes n'est pas équivalent à la contrainte WeightedSubCircuits au niveau du filtrage.

Proposition 1.

$$\text{WSC}[G](\{Set_k\}_{k=1..K}, Set_{dummy}, \{Next_i\}_{i=1..n}, \{Cost_k\}_{k=1..K}, Z) \equiv$$

$$\text{AllDifferent}(Next_1, ..., Next_n) \quad (2)$$

$$\wedge \quad \text{Partition}(Set_1, ..., Set_K, Set_{dummy}) \quad (3)$$

$$\wedge \quad \forall i \in V,\ i \in Set_{dummy} \Leftrightarrow Next_i = i \quad (4)$$

$$\wedge \quad \forall i \in V,\ \forall k = 1..K, dummy,\ i \in Set_k \Leftrightarrow Next_i \in Set_k \quad (5)$$

$$\wedge \quad \forall k = 1..K,\ \text{NoSubTours}(Set_k, Next_1, ..., Next_n) \quad (6)$$

$$\wedge \quad \forall k = 1..K,\ \textstyle\sum_{i \in Set_k} c(i, Next_i) = Cost_k \quad (7)$$

$$\wedge \quad \textstyle\sum_{k=1}^{K} Cost_k \leq Z \quad (8)$$

La première contrainte impose que deux variables successeurs ne peuvent pas avoir la même valeur.

La deuxième contrainte concerne uniquement les variables ensemblistes. Rappelons que chaque variable Set_k $\forall k \in 1..K$ représente l'ensemble des sommets de G présents dans le $k^{\text{ème}}$ circuit élémentaire (ou boucle) et la variable Set_{dummy} est l'ensemble des sommets exclus des circuits. Ainsi, chaque sommet est associé à un unique ensemble. La contrainte Partition utilisée ici n'est pas une partition au sens mathématique car elle autorise des ensembles vides.

Les contraintes de *channelling* (4) et (5) assurent la cohérence entre les variables ensemblistes Set_k et les variables de successeurs $Next_i$.

La contrainte (6) garantit que les sommets de chaque ensemble Set_k font partie d'un circuit élémentaire décrit par les variables de successeurs $Next_i$. Nous utilisons ici une version adaptée de la contrainte de Pesant *et al.* [12] vue plus haut. Notre contrainte interdit les affectations $Next_i = j$ si l'arc (i, j) provoque un sous-circuit de taille strictement inférieure à $|Set_k|$ dans le sous-graphe induit $G[Set_k]$.

La contrainte (7) garantit que le coût du circuit associé à chaque Set_k est majoré par la variable $Cost_k$. On peut définir une contrainte globale dont la portée sera (Set_k, $\{Next_i\}$, $Cost_k$) et dont le filtrage correspond à une somme partielle. La contrainte (8) garantit que la somme des coûts des circuits ne dépasse pas Z.

La proposition 1 permet de propager la contrainte WeightedSubCircuits en se basant sur des contraintes existantes ou des variantes relativement proches. Néanmoins, cette approche ne permettra pas d'atteindre le niveau de filtrage équivalent à celui de la contrainte WeightedCircuit. Dans la prochaine section nous montrerons comment améliorer ce filtrage en tenant compte des coûts des circuits.

4 Filtrage basé sur les coûts

Caseau et Laburthe [3] ont été les premiers à proposer un algorithme de filtrage basé sur le coût du circuit. Depuis, plusieurs travaux [6, 2] ont montré l'intérêt d'avoir ce filtrage complémentaire. À partir d'une instanciation partielle des variables, le but de cet algorithme est d'avoir une meilleure estimation de $LB(Cost_k)$, de $LB(Z)$ et de détecter certains arcs obligatoires ou interdits dans les circuits.

Les algorithmes de filtrage présentés dans cette section sont appliqués lorsque le graphe $G = (V, A, c)$ respecte la propriété *d'inégalité triangulaire faible* :

Propriété 1. *Nous dirons qu'un graphe $G(V, A, c)$ respecte l'*inégalité triangulaire faible *ssi $\forall (i, j) \in A$, $c(i, j) \leq$ coût de tout chemin entre i et j dans G.*

À la manière de la contrainte WeightedCircuit, nous cherchons une borne inférieure à $Cost_k$ pour tout $k = 1..K$. $Cost_k$ est la variable qui représente le coût du circuit hamiltonien dans $G[Set_k]$, le sous-graphe induit de G par les sommets présents dans Set_k. Par la suite, nous considérons seulement les ensembles tels que $|LB(Set_k)| > 1$, $\forall k = 1..K$. Suivant l'état de la variable ensembliste, on peut distinguer deux cas.

Premièrement, si Set_k est instanciée (c'est-à-dire $LB(Set_k) = UB(Set_k)$) alors tous les sommets de l'ensemble et uniquement eux participent au $k^{\text{ème}}$ circuit élémentaire de G. Le circuit est donc un circuit hamiltonien de $G[Set_k]$. Dans ce cas, on peut utiliser les mêmes relaxations que pour WeightedCircuit sur $G[Set_k]$ afin de déterminer une borne inférieure du coût du circuit.

Deuxièmement, si Set_k n'est pas instanciée alors les sommets de $UB(Set_k)$ sont partagés en deux catégories. Il y a les sommets de $LB(Set_k)$ forcément impli-

qués dans le circuit. Ces sommets sont dit *terminaux*. Les autres sont les sommets dit *potentiels* et appartiennent à $UB(Set_k) \backslash LB(Set_k)$.

On cherche ici une borne inférieure au coût du $k^{\text{ème}}$ circuit élémentaire en sachant que tous les sommets terminaux en font obligatoirement partie. Chercher ce circuit correspond au problème du circuit de Steiner présenté plus haut. La suite est consacrée à l'établissement d'une borne inférieure au coût du circuit de Steiner.

4.1 Calcul d'une borne inférieure au coût d'un circuit de Steiner

Comme le MSA pour le problème du voyageur de commerce, le problème de l'arborescence de Steiner (Directed Steiner Tree) est une relaxation du circuit de Steiner. Ce problème correspond à la recherche d'une arborescence de poids min dans un graphe enraciné telle que l'arborescence contient au moins tous les sommets terminaux. C'est une généralisation à la fois du MSA et du problème plus connu de l'arbre de Steiner (Steiner Tree Problem) dans les graphes non-orientés.

Malheureusement, les problèmes de l'arborescence et de l'arbre de Steiner sont NP-difficiles [13, 10]. Il n'est donc pas envisageable d'utiliser ces relaxations pour trouver une borne inférieure au circuit de Steiner dans notre algorithme de filtrage. On peut faire la même remarque pour le problème d'assignement.

Nous allons également montrer que raisonner uniquement sur le sous-graphe induit par les sommets terminaux est incorrect.

Proposition 2. *Soit $G = (V, A, c)$ un graphe respectant l'inégalité triangulaire faible et $V^* \subseteq V$. Le coût d'un MSA (ou AP) sur le sous-graphe $G[V^*]$ peut être supérieur au coût du circuit de Steiner couvrant dans G les sommets terminaux de V^*.*

Démonstration. Prenons par exemple le graphe de la figure 4 et $V^* = \{1, 2, 3, 4\}$. Le coût du MSA sur $G[V^*]$ vaut 22 alors que le coût du circuit de Steiner sur (G, V^*) vaut 21. □

Afin de pouvoir établir une borne inférieure du coût d'un circuit hamiltonien sur chaque Set_k, nous proposons de calculer un MSA (ou AP) dans un nouveau graphe orienté $G^* = (V^*, A^*, c^*)$.

Définition 1. *Soit $G = (V, A, c)$ un graphe orienté respectant l'inégalité triangulaire faible et $V^* \subseteq V$ un ensemble de sommets terminaux. Le* graphe extrapolé *$G^* = (V^*, A^*, c^*)$ est défini ainsi :*

- *si $(i, j) \in A$ alors $(i, j) \in A^*$ et $c^*(i, j) = c(i, j)$.*
- *sinon, s'il existe au moins un chemin entre i et j dans G composé exclusivement de sommets $\notin V^*$ (sauf i et j) alors $(i, j) \in A^*$ et $c^*(i, j)$ est égal au coût du plus court de ces chemin.*

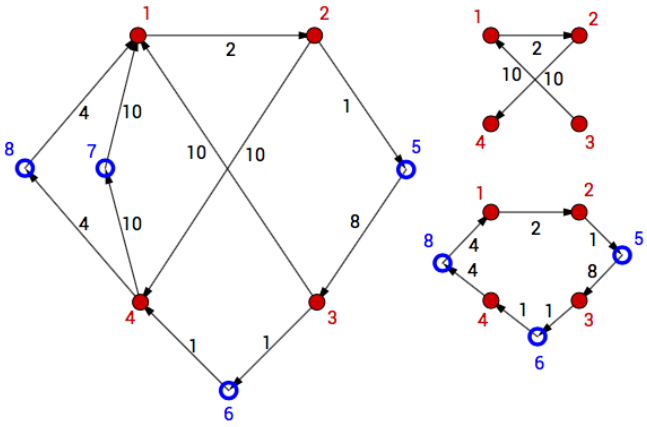


FIGURE 4 – Graphe avec des sommet terminaux en rouge (plein) et potentiels en bleu (cercle). En haut : la solution du MSA sur les sommets terminaux. En bas : la solution au circuit de Steiner sur le graphe entier.

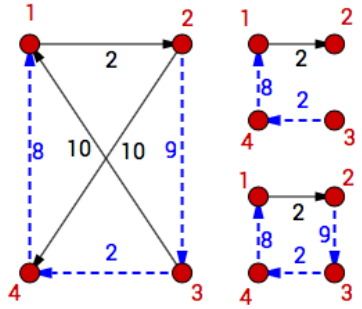


FIGURE 5 – Graphe G^* à partir du graphe de la figure 4. En haut, la solution au MSA et en bas la solution au AP sur G^*.

Exemple 3. *La figure 5 présente le* graphe extrapolé *correspondant au graphe de la figure 4. La transformation en G^* ajoute un arc $(4, 1)$ car il existe un chemin entre les sommets 4 et 1 $(4 - 8 - 1)$ de poids 8 (plus court chemin). Même chose pour $(3, 4)$ et $(2, 3)$ de poids respectifs 2 et 9.*

Nous allons montrer que le coût d'une solution au AP (ou MSA) dans le graphe G^* est une borne inférieure au coût du circuit de Steiner dans G pour V^*.

Lemme 1. *Étant donné $G = (V, A, c)$ respectant l'inégalité triangulaire faible et $G^* = (V^*, A^*, c^*)$ le graphe extrapolé défini sur $V^* \subseteq V$. Le coût d'un AP (resp. MSA) sur G^* est une borne inférieure au coût du circuit de Steiner dans G pour V^*.*

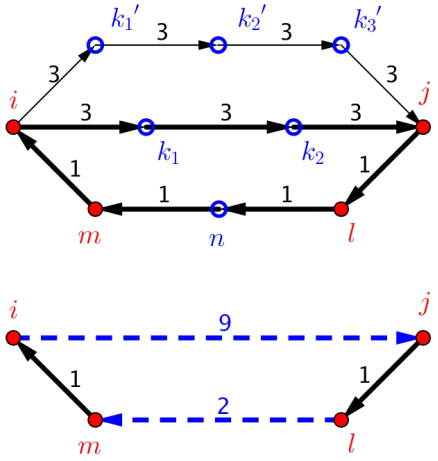


FIGURE 6 – En haut le graphe avec les sommets terminaux en rouge (plein) et potentiels en bleu (cercle). $\mathcal{C}$ est représenté par les arcs en gras. En bas, $\mathcal{C}^*$ du graphe extrapolé G^* sur les sommets terminaux. Le sous-chemin (i, k_1, k_2, j) de $\mathcal{C}$ est remplacé par l'arc (i, j) dans $\mathcal{C}^*$.

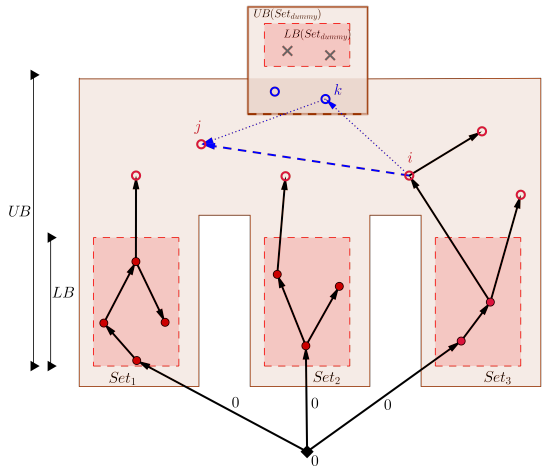


FIGURE 7 – Illustration du MSA sur $G^{*'}$. En rouge (plein et cercle) sont les sommets obligatoires. Les cercles bleus représentent les sommets potentiels et les cruciformes sont ignorés.

Démonstration. Soit $\mathcal{C}$, une solution optimale au circuit de Steiner dans G pour V^*. Comme le montre la figure 6, le circuit $\mathcal{C}$ comporte des sous-chemins de la forme $(i, k_1, ..., k_p, j)$ où $i, j \in V^*$ et $k_1, ..., k_p \in V \backslash V^*$. Par construction de G^*, il existe nécessairement un arc (i, j) dans A^* tel que $c^*(i, j) \leq (c(i, k_1) + \ldots + c(k_p, j))$. En remplaçant dans $\mathcal{C}$ tous les sous-chemins $(i, k_1, ..., k_p, j)$ par l'arc (i, j) correspondant dans A^*, on obtient un circuit hamiltonien de G^*. Notons $\mathcal{C}^*$ ce circuit.

En supprimant un arc de $\mathcal{C}^*$, on obtient une arborescence recouvrante de G^*de coût inférieur au coût de $\mathcal{C}^*$ (car tous les poids des arcs sont positifs). Le coût d'un MSA de G^* est donc inférieur au coût du circuit de Steiner dans G pour V^*.

De manière analogue, on peut remarquer que le cycle $\mathcal{C}^*$ est une solution au problème d' affectation (AP) sur le graphe G^*. □

Exemple 4. *Reprenons l'exemple 3. Dans G^*, le coût du MSA est de 11 et le coût d'un AP est de 21. Ces deux valeurs sont bien des bornes inférieures au cycle de Steiner dans le graphe G sur V^*.*

4.2 Application au filtrage des coûts des circuits

Pour filtrer la borne inférieure de $Cost_k$, nous pouvons calculer sur le graphe $G^* = (LB(Set_k), A^*, c^*)$ une des deux relaxations (MSA ou AP) pour avoir une minoration du coût du circuit. Comme le calcul d'un MSA ou d'un AP est polynomial tout comme la création de G^*, l'algorithme de filtrage proposé ici reste polynomial.

Comme pour la contrainte WEIGTHEDCIRCUIT [2, 6], disposer d'une bonne estimation du coût minimal de chaque circuit de Set_k permet aussi d'éliminer certains arcs incompatibles avec la borne supérieure de $Cost_k$. On peut ainsi filtrer des valeurs dans les domaines de certains $Next_i$ (pour $i \in LB(Set_k)$).

Intéressons nous maintenant à la variable Z qui correspond au coût global des circuits dans les Set_k.

La contrainte globale WEIGHTEDSUBCIRCUITS peut permettre un filtrage plus fin que celui réalisé uniquement au niveau du circuit de chaque Set_k. L'ensemble Set_{dummy} correspond aux sommets qui ne feront partie d'aucun circuit formé par un ensemble Set_k pour $k \leq K$. Parmi les sommets qui ne sont pas dans $UB(Set_{dummy})$, certains sont déjà affectés à un circuit et font partie d'un ensemble $LB(Set_k)$ $(k \leq K)$. Les autres sommets feront nécessairement partie d'un circuit mais ils ne sont pas pris en compte par les filtrages réalisés pour chaque circuit. Il est donc possible de déterminer une borne inférieure au coût total des futurs circuits en considérant l'ensemble des sommets qui ne seront pas dans Set_{dummy}. Notons $\mathcal{O}$ l'ensemble de ces sommets *obligatoires*. On a $\mathcal{O} = V \setminus UB(Set_{dummy})$.

Considérons le graphe $G' = (V \cup \{0\}, A', c')$ tel que $A \subset A'$ et A' contient un arc reliant le nouveau sommet 0 à un seul sommet de chaque ensemble $LB(Set_k)$, pour $k \leq K$. On se place ici dans le cas où $|LB(Set_k)| \geq 1, \forall k \leq K$. La fonction de coût c' est identique à c avec un coût nul pour tout arc issu de 0.

On construit le graphe extrapolé $G^{*\prime} = (\mathcal{O} \cup \{0\}, A^{*\prime}, c^{*\prime})$ sur G'. Comme précédemment, on peut montrer qu'un MSA de $G^{*\prime}$ est une borne inférieure à la somme des coûts des circuits élémentaires sur les Set_k.

Par exemple, la figure 7 illustre le MSA sur le graphe $G^{*\prime}$. Le graphe $G^{*\prime}$ est construit avec les sommets obligatoires déjà associés à un circuit (en rouge plein) et les sommets qui feront partie d'un circuit (en cercle rouge). Les sommets dans $UB(Set_{dummy})\backslash LB(Set_{dummy})$ (en cercle bleu) sont aussi pris en compte dans la construction : l'arc (i, j) est crée car le chemin (i, k, j) existe. Le sommet 0 et les arcs incidents de poids nul sont ajoutés.

5 Conclusion

Dans cet article, nous nous sommes intéressés aux contraintes de circuits qui sont notamment utilisées pour traiter des problèmes de voyageur de commerce (TSP) ou de tournées (VRP). Nous avons montré que les contraintes Circuit et WeigthedCircuit proposées dans la littérature ne permettent pas de modéliser tous les types de problèmes. Nous avons proposé une nouvelle contrainte nommée WeigthedSubCircuits qui permet de chercher un ensemble de circuits élémentaires disjoints sur un graphe valué en excluant certains sommets. Nous avons montré que même la cohérence aux bornes (BC) n'est pas atteignable en temps polynomial. Nous avons également montré que les méthodes efficaces proposées pour filtrer la contrainte WeigthedCircuit ne sont pas directement applicables à la nouvelle contrainte. Nous avons proposé une approche originale pour déterminer une borne inférieure aux coûts des circuits ainsi qu'au coût total de ces circuits. Notre approche repose sur une méthode originale pour déterminer une borne inférieure du coût d'un circuit de Steiner. Ces résultats permettent d'envisager plusieurs possibilités de filtrage pour la contrainte WeigthedSubCircuits.

Références

[1] Nicolas Beldiceanu and Evelyne Contejean. Introducing global constraints in CHIP. *math comp.*, 20(12) :97–123, 1994.

[2] 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.

[3] 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.

[4] Sylvain Ducomman, Hadrien Cambazard, and Bernard Penz. Alternative filtering for the weighted circuit constraint : Comparing lower bounds for the TSP and solving TSPTW. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 3390–3396, 2016.

[5] Filippo Focacci, Andrea Lodi, and Michela Milano. Cost-based domain filtering. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming - CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings*, volume 1713 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 1999.

[6] Filippo Focacci, Andrea Lodi, and Michela Milano. Embedding relaxations in global constraints for solving TSP and TSPTW. *Ann. Math. Artif. Intell.*, 34(4) :291–311, 2002.

[7] 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.

[8] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6) :1138–1162, 1970.

[9] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees : Part II. *Math. Program.*, 1(1) :6–25, 1971.

[10] Frank K Hwang, Dana S Richards, and Pawel Winter. *The Steiner tree problem*, volume 53. Elsevier, 1992.

[11] Latife Genç Kaya and John N Hooker. A filter for the circuit constraint. In *International Conference on Principles and Practice of Constraint Programming*, pages 706–710. Springer, 2006.

[12] 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.

[13] Dimitri Watel. *Approximation de l'arborescence de Steiner. (Approximation of the Directed Steiner Tree Problem).* PhD thesis, Versailles Saint-Quentin-en-Yvelines University, France, 2014.

Vers une stratégie de réduction de la base de clauses apprises fondée sur la relation de dominance

Jerry Lonlac[1,2] **Engelbert Mephu Nguifo**[1]

[1] Université Clermont Auvergne, CNRS, LIMOS, F-63000 Clermont-Ferrand, France
[2] Université Clermont Auvergne, CNRS, GEOLAB, F-63000 Clermont-Ferrand, France
lonlac@isima.fr, engelbert.mephu_nguifo@uca.fr

Résumé

L'apprentissage de clauses est l'une des composantes les plus importantes des solveurs SAT modernes CDCL (Conflict Driven Clause Learning), efficace sur les instances SAT industrielles. Sachant que le nombre de clauses apprises est prouvé être exponentiel dans le pire des cas, il est nécessairement d'identifier les clauses les plus pertinentes à maintenir et supprimer celles jugées non-pertinentes. Plusieurs stratégies de suppression des clauses apprises ont été proposées dans la littérature. Cependant, la diversité à la fois du nombre de clauses à supprimer à chaque étape de réduction et des résultats obtenus avec chaque stratégie constitue une difficulté de choix du meilleur critère. Ainsi, le problème du choix des clauses apprises à supprimer durant la recherche reste un véritable défi. Dans ce papier, nous proposons une nouvelle stratégie pour identifier les clauses apprises les plus pertinentes sans favoriser ou exclure une des mesures de pertinence proposées, mais en adoptant la notion de relation de dominance entre ces mesures. Notre approche s'affranchit du problème de la diversité des résultats et cherche un compromis entre les performances de ces mesures. De plus, l'approche proposée évite un autre problème non trivial qui est celui de déterminer le nombre de clauses à supprimer à chaque étape de réduction de la base des clauses apprises.

Abstract

Clause Learning is one of the more important components of conflict driven clause learning (CDCL) SAT solver that is effective on industrial instances. Since the number of learned clauses is proved to be exponential in the worse case, it is necessarily to identify the most relevant clauses to maintain and delete the irrelevant ones. As reported in the literature, several learned clauses deletion strategies have been proposed. However the diversity in both the number of clauses to be removed at each step of reduction and the results obtained with each strategy creates confusion to determine which criterion is better. Thus, the problem to select which learned clauses are to be removed during the search step remains very challenging. In this paper, we propose a novel approach to identify the most relevant learned clauses without favoring or excluding any of the proposed measures, but by adopting the notion of dominance relationship among those measures. Our approach bypasses the problem of the diversity of results and reaches to a compromise between the assessments of these measures. Furthermore, the proposed approach also avoids another non-trivial problem which is the amount of clauses to be deleted at each reduction of the learned clause database.

1 Introduction

Le problème SAT, i.e., le problème de décider si une formule booléenne sous forme normale conjonctive (CNF) est satisfiable ou non est central en Informatique et en Intelligence Artificielle incluant les problèmes de satisfaction de contraintes (CSP), planification, raisonnement non-monotone, vérification de logiciels, etc. Aujourd'hui, le problème SAT a gagné une audience considérable avec l'avénement d'une nouvelle génération de solveurs capables de résoudre des grandes instances issues du codage des applications du monde réel. Ces solveurs, communément appelés *solveurs SAT modernes* [15, 9] ou solveurs SAT CDCL (Conflict Driven Clause Learning) se sont avérés être très efficaces pour résoudre les instances SAT issues des applications du monde réel. Ils sont construits en intégrant quatre composants majeurs à la procédure *DPLL* classique [7] : les structures de données paresseuses [15], les heuristiques de choix de variables ba-

sées sur les activités (comme VSIDS), [15], les stratégies de redémarrage [11], et l'apprentissage de clauses [18, 15]. Bien qu'une combinaison astucieuse de ces composants contribue à améliorer l'efficacité des solveurs SAT modernes [14], l'apprentissage de clauses reste connu comme le composant le plus important [16].

L'idée principale de l'apprentissage de clauses est qu'au cours du processus de propagation unitaire, quand une branche courante de l'arbre de recherche méne à un conflit, les solveurs SAT modernes apprennent une clause conflit qui aide la propagation unitaire à découvrir l'une des implications manquées à un niveau plus haut de l'arbre de recherche. Cette clause conflit exprime les causes du conflit et est utilisée pour élaguer l'espace de recherche. L'apprentissage de clauses, aussi connu dans la littérature comme *"Conflict Driven Clause Learning" (CDCL)* se réfère maintenant au schéma d'apprentissage le plus connu et utilisé premier "UIP" ("Unique Implication Point") [21], initialement intégré dans le solveur SAT *Grasp* [17] et efficacement mis en oeuvre dans *zChaff* [15]. La plupart des solveurs SAT intégrent ce solide schéma d'apprentissage. Comme à chaque conflit, les solveurs CDCL apprennent une nouvelle clause qui est ajoutée à la base des clauses apprises, et que le nombre de clauses apprises se rèvèle être exponentiel dans le pire des cas, il est nécessaire de supprimer certaines clauses apprises afin de maintenir une base de clauses de taille polynomiale. Cependant, supprimer trop de clauses peut rendre l'apprentissage inefficace, et garder trop de clauses également peut altérer l'efficacité de la propagation unitaire. Il est nécessaire d'identifier les clauses apprises les plus pertinentes à maintenir pour la suite de la recherche.

La gestion de la base des clauses apprises a fait l'objet de plusieurs études et plusieurs stratégies de gestion de la base des clauses apprises furent proposées dans la littérature [15, 17, 9, 2, 3, 12]. Ces stratégies furent proposées avec pour objectif de maintenir une base de clauses apprises de taille raisonnable en éliminant les clauses jugées non pertinentes pour la suite de la recherche. Le principe général de ces stratégies est que, à chaque conflit, une activité est associée à la clause apprise (stratégie statique). Cette activité basée sur une heuristique, vise à pondérer chaque clause en fonction de sa pertinence dans le processus de recherche. Dans le cas des stratégies dynamiques, cette activité est mise à jour dynamiquement. La réduction de la base des clauses apprises consiste à éliminer les clauses inactives ou non pertinentes. Bien que toutes les stratégies de suppression de clauses apprises proposées dans la littérature se révèlent empiriquement efficaces, identifier les clauses les plus pertinentes à maintenir pour le processus de recherche reste une tâche difficile. Notre motivation pour ce travail vient de l'observation que l'utilisation de différentes stratégies de suppression fondées sur la pertinence donne des performances différentes. Notre but est de tirer profit de plusieurs stratégies de suppression de clauses apprises.

Dans ce papier, nous intégrons un point de vue basé sur les préférences de l'utilisateur dans le processus de résolution SAT. A cet effet, nous intégrons dans le processus SAT l'idée des requêtes *skyline* [5], *motifs dominants* [19], *règles d'association non dominées* [6] afin d'apprendre des clauses de façon à se libérer de la contrainte du nombre de clauses apprises à supprimer à chaque réduction de la base des clauses apprises. De telles requêtes ont retenu beaucoup d'attention en raison de leur importance dans la prise de décision multi-critères. Etant donné un ensemble de clauses, l'ensemble `skyline` contient les clauses qui ne sont dominées par aucune autre clause.

Le traitement `skyline` ne réquiert aucune fonction de sélection de seuil, et la propriété formelle de dominance satisfait par les clauses `skyline` donne aux clauses un intérêt global avec des sémantiques facilement comprehensibles par l'utilisateur. Cette notion de `skyline` a été développée pour des applications de base de données et de fouille de données, cependant elle n'a jamais été utilisée pour SAT. Dans ce papier, nous adaptons cette notion au processus de gestion des clauses apprises.

Le papier est organisé comme suit : nous présentons quelques stratégies efficaces de suppression de clauses apprises fonées sur la pertinence utiliséss dans la littérature. Ensuite, notre stratégie de suppression de clauses apprises fondée sur la relation de dominance entre différentes stratégies est présentée dans la section 3. Finalement, avant de conclure, les résultats expérimentaux démontrant l'efficacité de notre approche sont présentés.

2 Sur les stratégies de gestion de la base des clauses apprises

Dans cette section, nous présentons quelques mesures de qualité des clauses apprises exploitées par la plupart des solveurs SAT de la littérature.

Le solveur SAT CDCL le plus populaire *Minisat* [9] considère comme pertinentes les clauses les plus impliquées dans les récentes analyses de conflits et élimine les clauses apprises dont l'implication dans les récentes analyses de conflits est marginale. Une autre stratégie appelée *LBD* pour *Literal Block Distance* fut proposée dans [2]. La mesure *LBD* dont l'efficacité a été prouvée empiriquement est aussi exploitée par la plupart des meilleurs solveurs SAT de l'état de l'art (*Glu-*

cose, Lingeling [4]). Cette mesure utilise le nombre de niveaux différents impliqués dans une clause apprise donnée pour quantifier la qualité de la clause apprise. Ainsi, les clauses apprises avec les plus petites valeurs de *LBD* sont considérées comme les plus pertinentes. Dans [3], une nouvelle police de gestion dynamique de la base des clauses apprises est proposée. Elle est fondée sur un principe d'activation et de gel dynamique des clauses apprises. Elle active les clauses apprises les plus prometteuses tout en gelant celles jugées non pertinentes.

Dans [12], un nouveau critère pour quantifier la pertinence d'une clause apprise en utilisant son niveau de retour-arrière appelé *BTL* pour `BackTrack Level` est proposé. A partir des expérimentations, les auteurs observent que les clauses apprises avec les petites valeurs de *BTL* sont plus souvent utilisées dans le processus de propagation unitaire que celles ayant des grandes valeurs *BTL*. Plus précisément, les auteurs observent que les clauses apprises ayant les valeurs *BTL* inférieures à 3 sont toujours plus utilisées dans la propagation unitaire que le reste des clauses. A partir de cette observation, et motivés par le fait qu'une clause apprise avec une petite valeur *BTL* contient plus de littéraux du haut de l'arbre de recherche, les auteurs déduisent que les clauses apprises les plus pertinentes sont celles permettant un retour-arrière plus haut dans l'arbre de recherche (clauses ayant de petites valeurs *BTL*).

Plus récemment, plusieurs autres stratégies de gestion de la base des clauses apprises furent proposées dans [13, 1]. Dans [13], les auteurs explorent un nombre varié de stratégies de réduction de la base des clauses apprises, et les performances des différentes extensions du solveur *Minisat* intégrant leurs stratégies sont evaluées sur les instances SAT prises des compétitions SAT 2013 et 2014, et comparées aux autres solveurs SAT de la littérature (*Glucose, Lingeling*) ainsi qu'au solveur original *Minisat*. A partir des performances obtenues avec leurs stratégies, les auteurs de [13] ont montré que les stratégies d'apprentissage fondées sur sur la mise en place d'une borne supérieure sur la taille des clauses et proposées il y a plus d'une quinzaine d'années restent de bonnes mesures pour prédire la qualité des clauses apprises. Ils montrent que l'ajout de la randomisation à l'apprentissage fondée sur la mise en place d'une borne supérieure sur la taille des clauses est un bon moyen de parvenir à une diversification contrôlée, et permet de favoriser les clauses de petites tailles tout en maintenant une petite fraction de clauses de grande taille nécessaires pour la dérivation des preuves par résolution sur certaines instances difficiles. Cette étude ouvre plusieurs discussions autour des stratégies proposées pour la gestion des clauses apprises et soulève des questions sur l'efficacité proclamée par d'autres stratégies de l'état de l'art [9, 2].

Dans [1], les auteurs utilisent la structure communautaire des instances SAT industrielles pour identifier un ensemble de clauses apprises fortement utiles. Ils montrent que l'augmentation d'une instance SAT avec les clauses apprises par le solveur pendant son exécution n'est pas toujours un moyen de rendre l'instance facile à résoudre. Les auteurs montrent cependant qu'en ajoutant à l'instance originale un ensemble de clauses fondé sur la structure communautaire de l'instance améliore les performances du solveur dans plusieurs cas.

Les différentes performances obtenues par chaque stratégie suggèrent que la question de prédire efficacemment les meilleures clauses apprises à maintenir pour la suite de la recherche est toujours ouverte et mérite de fortes investigations.

D'autre part, il est important de noter que l'efficacité de la plupart de ces stratégies de gestion des clauses apprises de létat de l'art depend fortement de la fréquence de nettoyage et de la quantité de clauses à supprimer à chaque nettoyage. Généralement, tous les solveurs SAT CDCL utilisant ces stratégies suppriment exactement la moitié des clauses apprises à chaque étape de nettoyage. Par exemple, les solveurs SAT CDCL *Minisat* [9] et *Glucose* [2] suppriment la moitié des clauses apprises à chaque étape de nettoyage. Cependant, l'efficacité de cette quantité de clauses à supprimer (c'est à dire la moitié) à chaque étape de nettoyage de la base de clauses apprises n'a pas été démontrée théoriquement, mais plutôt expérimentalement. A notre connaissance, il n'existe pas beaucoup d'étude dans la littérature sur comment déterminer la quantité de clauses à supprimer à chaque nettoyage. Ce papier propose une approche pour identifier les clauses apprises les plus pertinentes au cours du processus de résolution sans favoriser une des mesures de qualité de la littérature. Cette approche s'affranchit du problème de la determination de la quantité de clauses à supprimer à chaque nettoyage : la quantité de clauses apprises à supprimer correspond à chaque fois au nombre de clauses apprises dominées par une clause particulière de l'ensemble de clauses apprises courant qui est appelée dans les sections suivantes, `clause apprise de référence`.

3 Détection des clauses apprises non dominées

Nous présentons maintenant notre mesure de pertinence des clauses apprises fondée sur la relation de dominance. Tout d'abord, nous motivons cette approche à partir d'un simple exemple, et proposons ensuite un algorithme permettant d'identifier les clauses apprises

les plus pertinentes, ainsi que quelques détails techniques.

3.1 Motivation

Considérons les stratégies de pertineneces suivantes : *LBD* [2], *SIZE* (stratégie qui considère comme pertinentes les clauses de petites tailles) et la mesure de pertinence utilisée par *minisat* [9] que nous appelons ici *CVSIDS*. Supposons que nous ayons dans la base des clauses apprises, les clauses c_1, c_2 et c_3 avec :

- $SIZE(c_1) = 8$, $LBD(c_1) = 3$, $CVSIDS(c_1) = 1e^{100}$,
- $SIZE(c_2) = 6$, $LBD(c_2) = 5$, $CVSIDS(c_2) = 1e^{200}$,
- $SIZE(c_3) = 5$, $LBD(c_3) = 4$, $CVSIDS(c_3) = 1e^{300}$.

La queston que nous posons est la suivante : laquelle des clauses est pertinente ? Dans [2], les auteurs considèrent la clause c_1 qui a le plus petit *LBD* comme la plus pertinente. Au contraire, les auteurs de [13] et [10] préfèrent la clause c_3 tandis que la préférence des auteurs de *Minisat* [9] va à la clause c_3. Notre approche s'affranchit de cette préférence particulière à une mesure en cherchant un compromis entre les différentes mesures de pertinence à travers une relation de dominance. Ainsi, pour la situation décrite plus haute, seulement la clause c_2 est non pertinente, elle est dominée par la clause c_3 sur les trois mesures données.

3.2 Formalisation

Au cours de la recherche, les solveurs SAT CDCL apprennent un ensemble de clauses qui sont stockées dans une base des clauses apprises Δ, $\Delta = \{c_1, c_2, ..., c_n\}$. A chaque étape de nettoyage, nous évaluons ces clauses sur un ensemble de mesures de pertinence $\mathcal{M} = \{m_1, m_2, ..., m_k\}$. Nous notons $m(c)$ la valeur de la mesure m pour la clause c, $c \in \Delta$, $m \in \mathcal{M}$. Comme l'évaluation des clauses apprises varie d'une mesure à une autre, l'utilisation de plusieurs mesures pourrait conduire à différents résultats (clauses pertinentes par rapport à une mesure de pertinence). Par exemple, si nous considérons l'exemple ci-dessus, c_1 est la meilleure clause par rapport à la mesure *LBD*, ce qui n'est pas le cas par rapport à la mesure d'évaluation *SIZE* qui favorise c_3. Cette différence des évaluations rend difficile le choix d'une mesure lors du processus de selection des clauses apprises. Ainsi, nous pouvons utiliser la notion de dominance entre clauses apprises afin de selectionner les clauses les plus pertinentes. Avant la formulation de la relation de dominance entre clauses apprises, il est nécessaire de la définir au niveau des valeurs des mesures de pertinence. Pour ce faire, nous définissons la `valeur de dominance` comme suit :

Définition 1 (Valeur de dominance) *Etant donnée une mesure de pertinence de clauses apprises m et deux clauses apprises c et c', nous disons que $m(c)$ domine $m(c')$, et notons $m(c) \succeq m(c')$, si et seulement si $m(c)$ est préférée à $m(c')$. Si $m(c) \succeq m(c')$ et $m(c) \neq m(c')$ alors nous disons que $m(c)$ domine strictement $m(c)'$ et notons $m(c) \succ m(c')$.*

Définition 2 (Dominance entre clauses) *Etant données deux clauses apprises c et c', la relation de dominance entre c et c' par rapport à un ensemble de mesures de pertinence $\mathcal{M}$ est définie comme suit :*

- *c domine c', noté $c \succeq c'$, si et seulement si $m(c) \succeq m(c')$, $\forall m \in \mathcal{M}$.*
- *si c domine c' et $\exists m \in \mathcal{M}$ tel que $m(c) \succ m(c')$, alors c domine strictement c' et nous notons $c \succ c'$.*

Pour découvrir les clauses apprises pertinentes, une approche naïve consiste à comparer chaque clause apprise avec toutes les autres clauses apprises. Cependant, le nombre de clauses apprises est prouvé être exponentiel, ce qui rend les comparaisons par paire très coûteuses. Dans la suite, nous montrons comment surmonter ce problème en définissant à chaque étape de nettoyage de la base de clauses apprises, une clause apprise particulière notée τ que nous appelons ici *clause apprise de référence* qui est une clause non dominée de la base des clauses apprises Δ relativement à l'ensemble des mesures de pertinence de clauses apprises $\mathcal{M}$. A chaque étape de nettoyage de la base des clauses apprises, toutes les clauses apprises dominées par τ seront considérées comme étant non pertinentes et seront ainsi supprimées de la base des clauses apprises.

Pour définir la *clause apprise de référence*, nous avons besoin de définir une nouvelle mesure de qualité fondée sur toutes les mesures de qualité des clauses apprises de $\mathcal{M}$. Nous appelons cette nouvelle mesure *"Dégré de compromis"*, en court *DegComp*. Elle est définie comme suit :

Définition 3 (Degré de compromis) *Etant donnée une clause apprise c, le degré de compromis de c par rapport à l'ensemble des mesures de pertinence de clauses apprises $\mathcal{M}$ est défini par $DegComp(c) = \frac{\sum_{i=1}^{n} \widehat{m_i(c)}}{|M|}$, où $\widehat{m_i(c)}$ correspond à la valeur normalisée de la mesure m_i pour la clause c.*

En effet, en pratique, les mesures sont hétérogènes et sont définies à différentes échelles. Par exemple, les valeurs de la mesure de qualité des clauses apprises dans [9] sont très grandes, alors que les valeurs de la mesure dans [2] sont petites. Ainsi, afin d'éviter que des mesures avec des grandes valeurs rendent marginales les mesures avec de petites valeurs dans le calcul du dégré de compromis d'une clause apprise donnée, il est nécessaire de normaliser les valeurs des différentes mesures de pertinence. Dans notre cas, nous

choisissons de normaliser toutes les mesures dans l'intervalle $[0,1]$. Plus précisément, chaque valeur de mesure $m(c)$ de toute clause apprise c doit être normalisée dans $[0,1]$. La normalisation d'une mesure donnée m est effectuée en fonction du domaine de ces valeurs et de la distribution statistique de son domaine actif. Nous rappelons que le domaine actif d'une mesure m est l'ensemble de ses valeurs possibles. Il convient de mentionner que la normalisation d'une mesure ne modifie pas la relation de dominance entre clauses. Si nous considérons la clause apprise c_1 donnée dans l'exemple de motivation à la section 3.1, avec ses valeurs sur les trois mesures considérées : $DegComp(c_1) = \frac{\widehat{CVSDIS}(c_1)+\widehat{LBD}(c_1)+\widehat{SIZE}(c_1)}{3}$, alors, nous avons $DegComp(c_1) = \frac{\frac{1}{1e^{100}}+\frac{3}{nVars()}+\frac{8}{nVars()}}{3}$, avec $nVars()$ le nombre de variables de la formule booléenne considérée.

Apès avoir donné les définitions néecessaires (*clause apprise de référence, dégré de compromis*), le lemme suivant offre une solution plus rapide que des comparaisons par paires pour rechercher les clauses apprises pertinentes fondées sur la relation de dominance.

Lemme 1 *Soit c une clause apprise ayant le dégré de compromis minimal en considérant l'ensemble des mesures de pertinence des clauses apprises $\mathcal{M}$, alors c est une clause apprise non dominée.*

Preuve 1 *Soit c une clause apprise ayant le dégré de compromis minimal en considérant l'ensemble des mesures de pertinence des clauses apprises $\mathcal{M}$, supposons qu'il existe une clause apprise c' qui domine strictement c, cela signifie que $\forall m \in \mathcal{M}$, $m(c') \succeq m(c)$ et $\exists m' \in \mathcal{M}$, $m'(c') \succ m'(c)$. Ainsi, nous avons $DegComp(c') < DegComp(c)$. La dernière inégalité contredit notre hypothèse de départ, puisque la clause c a le dégré de compromis minimal.*

Propriété 1 *Soit $\mathcal{M}$ l'ensemble des mesures de pertinence des clauses apprises, $\forall c, c', c"$ trois clauses apprises, si $c \succ c'$ et $c' \succ c"$ alors $c \succ c"$.*

Au cours de la recherche, à chaque étape de nettoyage de la base des clauses apprises, nous cherchons d'abord la clause apprise $cMin$ ayant le dégré de compromis minimal par rapport à l'ensemble des mesures de pertinence consodérées $\mathcal{M}$. Nous supprimons ensuite de la base des clauses apprises toutes les clauses dominées par $cMin$.

La recherche de toutes les clauses apprises non dominées au cours de chaque étape de nettoyage peut prendre beaucoup de temps, de ce fait, nous recherchons uniquement les clauses non dominées par la clause apprise de référence lors de chaque étape de nettoyage.

3.3 Algorithme

Dans cette sous-section, après avoir présenté le schéma général d'une stratégie de suppression des clauses apprises ($reduceDB(\Delta)$) adoptée par la plupart des solveurs SAT de la littérature, nous proposons un algoritme permettant de découvrir les clauses apprises pertinentes en utilisant la relation de dominance.

L'algorithme 1 présente le schéma général d'une stratégie de suppression des clauses apprises ($reduceDB(\Delta)$). Cet algorithme trie dans un premier temps l'ensemble des clauses apprises sur le critère défini et supprime ensuite la moitié des clauses apprises. En effet, cet algorithme prend une base de clauses apprises de taille n et retourne en sortie une base de clauses apprises de taille $\frac{n}{2}$. Ceci est différent de notre approche qui recherche prémièrement la clause apprise ayant le plus petit dégré de compromis (appelé `clause apprise de référence`) et supprime ensuite toutes les clauses apprises qu'elle domine.

L'algorithme 2 présente notre stratégie de suppression des clauses apprises. Il est important de noter que les clauses apprises dont la taille (en nombre de littéraux) et le `LBD` sont plus petits ou égaux à 2 ne sont pas concernées par la relation de dominance. Ces clauses apprises sont considérées comme pertinentes et sont maintenues dans la base des clauses apprises. Ainsi, la fonction $minDegComp$ de l'algorithme 2 cherche la clause apprise de dégré de compromis minimal parmi les clauses apprises de taille et de `LBD` supérieur à 2.

Algorithm 1: Stratégie de suppression : fonction `reduceDB`

Données: Δ : La base des clauses apprises de taille n
Résultat: Δ : La nouvelle base des clauses apprises de taille $n/2$

```
1  sortLearntClauses() ;      /* sur le critère défini */
2  limit = n/2;
3  ind = 0;
4  pour ind < limit faire
5      clause = Δ[ind] ;
6      si clause.size() > 2 && clause.lbd() > 2 alors
7          removeClause() ;
8      sinon
9          saveClause() ;
10     ind++;
11 retourner Δ ;
```

Algorithm 2: `reduceDB-Dominance-Relationship`

Données: Δ : La base des clauses apprises ; $\mathcal{M}$: un ensemble de mesures de pertinence
Résultat: Δ : La nouvelle base des clauses apprises

```
cMin = minDegComp(M) ;   /* cMin la clause ayant le dégré de compromis minimal par rapport à M */
ind = 0;
pour ind < |Δ| faire
    c = Δ[ind] ;        /* une clause apprise */
    si c.size() > 2 && c.lbd() > 2 && dominates(cMin, c, M) alors
        removeClause() ;
    sinon
        saveClause() ;
    ind++;
retourner Δ ;

Fonction dominates(cMin : une clause, c : une clause, M)
i = 0;
pour i < |M| faire
    m = M[i] ; /* une mesure de pertinence */
    si m(c) ⪰ m(cMin) alors
        retourner FAUX ;
    i++;
retourner VRAI ;
```

4 Expérimentationss

Pour nos expérimentations, nous utilisons trois mesures de pertinence pour la relation de dominance afin d'évaluer l'efficacité de notre approche. Il faut noter que l'utilisateur peut choisir de combiner différentes autres mesures de pertinence. Nous utilisons pour notre étude, les mesures SIZE [10], LBD [2] et $CVSIDS$ [9]. L'efficacité de toutes ces mesures a été prouvée dans la littérature [9, 2, 13]. Il est possible d'utiliser plus de mesures de pertinence dans la dominance, il convient de noter qu'en ajoutant une mesure à $\mathcal{M}$, le nombre de clauses apprises pertinentes maintenues peut décroître ou croître. La décroissance peut être expliquée par le fait qu'une clause apprise peut être dominée par rapport à un ensemble de mesures $\mathcal{M}$ et non dominée par rapport à un ensemble de mesure $\mathcal{M}'$, tel que $\mathcal{M} \subset \mathcal{M}'$. Par exemple, si deux clauses apprises c et c' sont non dominées par rapport à un ensemble de mesures $\mathcal{M}$, il est possible qu'une des deux clauses domine l'autre en supprimant une mesure. La croîssance peut être expliquée par le fait qu'une clause peut être dominée par rapport à $\mathcal{M}$ et non dominée par rapport à $\mathcal{M}'$. Par exemple, considérons une clause apprise c qui domine une autre clause apprise c' par rapport à un ensemble de mesures $\mathcal{M}$, en ajoutant une mesure de pertinence m à $\mathcal{M}$ telle que $m(c') \succ m(c)$, alors c' ne sera plus dominée par c.

Nous exécutons les différents solveurs SAT sur les 300 instances SAT prises de la dernière SAT-RACE 2015 et sur les 300 instances SAT prises de la dernière compétition SAT 2016. Toutes les instances sont prétraitées par `SatElite` [8] avant l'exécution du solveur SAT. Les expérimentations sont ménées sur une machine Quad-Core Intel XEON avec 32GB de mémoire fonctionnant à 2.66 Ghz. Pour chaque instance, nous utilisons un temps limite égal à 3600 secondes du temps CPU pour les instances de la SAT-RACE, et 10000 secondes pour celles de la compétition SAT 2016. Nous implémentons notre approche dans le solveur SAT *Glucose* et faisons une comparaison entre le solveur original et celui amélioré avec la nouvelle stratégie de suppression des clauses apprises qui utilise la relation de dominance et que nous appelons ici `DegComp-Glucose`.

4.1 Nombre d'instances résolues et temps CPU

Le tableau 1 présente les résultats obtenus sur les instances de la SAT-RACE 2015. Nous utilisons le code source de *Glucose* 3.0 avec la mesure *LBD* (écrit *LBD-Glucose* ou *Glucose* dans la suite). Nous remplaçons ensuite la mesure *LBD* par chacune des autres mesures de pertinence considérées ici, à savoir : $SIZE$-*Glucose* qui considère les clauses apprises de petites tailles comme les plus pertinentes, $CVSIDS$-*Glucose* qui maintient les clauses apprises les plus impliquées dans les récentes analyses de conflits, et finalement notre approche *DegComp-Glucose*. Le tableau 1 montre une évaluation expérimentale comparative des quatre mesures, ainsi que de *Minisat* 2.2. Dans la deuxième colonne du tableau 1, nous donnons le nombre total d'instances résolues (#Solved). Nous mentionnons aussi, le nombre d'instances prouvées satisfiables (#SAT) et le nombre d'instances prouvées insatisfiables (#UNSAT) entre parenthèses. La troisième colonne indique le temps CPU moyen en secondes (le temps total sur les instances résolues divisé par le nombre d'instances résolues). Sur la SAT-RACE 2015, notre approche *DegComp-Glucose* est plus efficace que toutes les autres approches en terme de nombre d'instances résolues (voir aussi la figure 1). En effet, le solveur original *Glucose* résout 236 instances, alors que le solveur amélioré avec notre approche de dominance résout 12 instances de plus. En effet, résoudre un tel nombre d'instances en plus est

clairement significatif pour la résolution pratique de SAT. Le solveur *CVSIDS-Glucose* résout 4 instances de plus que *Glucose* 3.0. *Minisat* 2.2 est le moins bon des solveurs parmi les 5 solveurs.

Solveurs	#Solved (#SAT - #UNSAT)	Temps Moyen
Minisat 2.2	209 (134 - 75)	585.19 s
SIZE-Glucose	230 (131 - 99)	533.86 s
CVSIDS-Glucose	240 (140 - 100)	622.23 s
LBD-Glucose	236(136 - 100)	**481.66 s**
DegComp-Glucose	**248 (146 - 102)**	571.31 s

TABLE 1 – Evaluation comparative des solveurs sur la SAT-RACE-2015.

Le tableau 2 montre 5 instances parmi les instances de la SAT-RACE 2015 résolues par notre approche et non résolues par par les trois autres approches (*LBD-Glucose*, *SIZE-Glucose*, *CVSIDS-Glucose*). Le temps utilisé pour résoudre ces instances peut expliquer en partie l'augmentation du temps moyen de résolution de *DegComp-Glucose*. En outre, nous constatons également qu'il n'y a aucune instance résolue par tous les autres solveurs et non résolue par notre approche (comme détaillé plus loin). Cela montre d'une part que l'application de la domination entre différentes mesures de pertinence ne dégrade pas les performances de ces mesures, mais tire profit de la performance de chacune des mesures, en considérant l'ensemble d'instances de la SAT-RACE 2015.

Instances	LBD	SIZE	CVSIDS	DegComp
jgiraldezlevy.2200.9086.08.40.8	-	-	-	**93.71 s**
manthey_DimacsSorterHalf_37_3	-	-	-	**2642.88 s**
14packages-2008seed.040	-	-	-	**1713.46 s**
manthey_DimacsSorter_37_3	-	-	-	**2673.39 s**
jgiraldezlevy.2200.9086.08.40.2	-	-	-	**3195.03 s**

TABLE 2 – Instances de la SAT-RACE 2015 résolues par *DegComp-Glucose* et non résolues par les autres solveurs.

La figure 1 montre les résultats des temps cumulés i.e le nombre d'instances (axe-x) résolues en un temps donné en secondes (axe-y). Cette figure donne pour chaque technique, le nombre d'instances (*#instances*) en t secondes. Elle confirme l'efficacité de notre approche fondée sur la relation de dominance. A partir de cette figure, nous pouvons observer que *DegComp-Glucose* est généralement plus rapide que tous les autres solveurs, même si le temps moyen de résolution de *LBD-Glucose* est le plus bas (voir le tableau 1). Bien que *DegComp-Glucose* ait besoin de plus de temps pour appliquer la relation de dominance, la qualité des clauses apprises restantes (sur la SAT-RACE) contribue à améliorer le temps nécessaire pour résoudre les instances.

Le tableau 3 présente les résultats sur les instances de la compétition SAT 2016. Ici, *LBD-Glucose* et *CVSIDS-Glucose* résolvent une instance de plus que *DegComp-Glucose* qui reste tout même compétitif, et

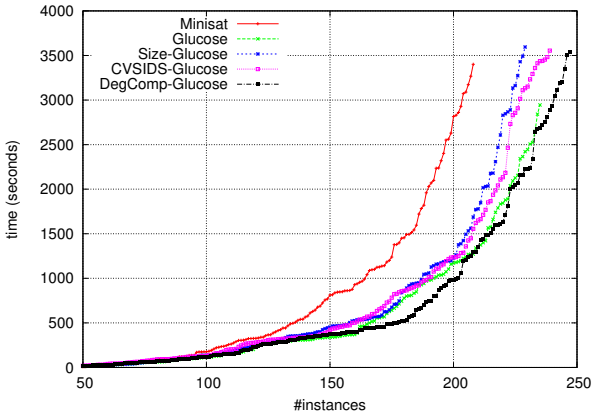


FIGURE 1 – **Evaluation des solveurs sur la SAT-RACE-2015**

Solveurs	#Solved (#SAT - #UNSAT)	Average Time
Minisat 2.2	138 (65 - 73)	1194.85 s
SIZE-Glucose	156 (67 - 89)	1396.73 s
CVSIDS-Glucose	165 (67 - **98**)	1368.99 s
LBD-Glucose	165 (68 - 97)	**1142.33 s**
DegComp-Glucose	164 (**69** - 95)	1456.34 s

TABLE 3 – Evaluation comparative des solveurs sur la Compétition SAT 2016.

résout un plus grand nombre d'instances satisfiables. La figure 2 présente les résultats des temps cumulés sur les instances de la compétition SAT 2016. Il ressort de ce deuxième jeu de données que *LBD-Glucose* est plus efficace que les autres, y compris le solveur intégrant notre approche qui reste compétitive par rapport au nombre d'instances résolues.

Ce résultat donne du crédit au théorème du "No Free Lunch" [20]. Nous pensons également que la fonction d'aggrégation ne peut pas être unique pour tous les jeux de données, de ce fait qu'il est nécessaire d'explorer la combinaison efficace des mesures préférées.

4.2 Instances communes résolues

Dans le tableau 4, l'intersection entre deux mesures de pertinece donne le nombre d'instances communes résolues par chaque mesure. Par exemple, *LBD* et *SIZE* résolvent 219 instances en commun, tandis que 234 instances sont résolues par *LBD* et *DegComp*. Nous pouvons voir que notre approche résout en commun le plus grand nombre d'instances avec chacune des mesures aggrégées. Plus précisément, pour chacune des mesures, le nombre d'instances résolues en commun avec une autre mesure est plus petit que le nombre d'instances résolues en commun avec notre approche.

Pour plus de détails, le tableau 5 donne le nombre d'instances communes résolues par les mesures de

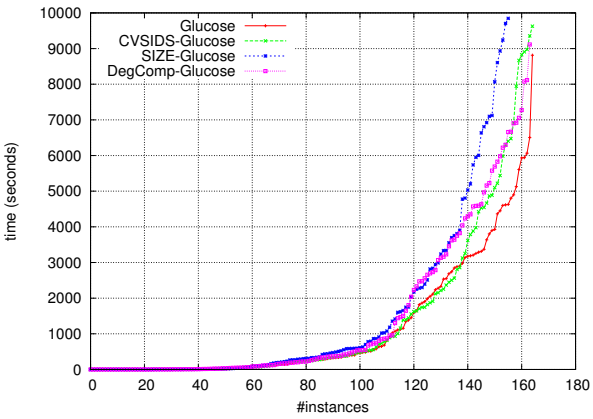


FIGURE 2 – **Evaluation des solveurs sur la compétition SAT 2016**

Mesures	LBD	SIZE	CVSIDS	DegComp
LBD	**236**			**234**
SIZE	219	**230**		**225**
CVSIDS	233	221	**240**	**238**
DegComp				**248**

TABLE 4 – Instances résolues communément sur la SAT-RACE 2015.

pertinence considérées. Ce tableau permet de voir le nombre d'instances communes résolues par 1, 2, 3 ou 4 mesures. Par exemple, les 4 stratégies utilisées dans l'expérimentation de notre approche résolvent en commun 218 instances, tandis que 44 instances ne sont résolues par aucune des stratégies. Nous pouvons observer que 1, 1, 5, et 5 sont respectivement les nombres d'instances résolues uniquement par LBD and $CVSIDS$, $SIZE$ et $DegComp$. De plus, il n'existe aucune instance résolue en commun par les trois stratégies (LBD, $SIZE$ et $CVSIDS$) et non résolue par notre approche $DegComp$.

Mesures		DegComp		¬DegComp	
		CVSIDS	¬CVSIDS	CVSIDS	¬CVSIDS
LBD	SIZE	218	1	**0**	0
	¬SIZE	1	1	1	1
¬ LBD	SIZE	3	3	0	**5**
	¬SIZE	3	**5**	1	44

TABLE 5 – Détails des instances communes résolues sur la SAT-RACE 2015.

4.3 Mesures combinées

Le tableau 6 donne le nombre d'instances résolues avec notre approche de dominance respectivement en fonction du nombre de mesures utilisées dans la relation de dominance. De ce tableau, nous pouvons voir que le nombre d'instances résolues en utilisant 2 mesures (au lieu de 3) dans la relation de dominance est toujours plus petit que le nombre d'instances résolues (248) en utilisant les 3 mesures.

Mesures	LBD	SIZE	CVSIDS	DegComp
LBD	**236**			
SIZE	223	**230**		
CVSIDS	239	242	**240**	
DegComp				**248**

TABLE 6 – Combinaison de deux mesures sur la SAT-RACE 2015.

4.4 Pourcentage de clauses supprimées

Au cours de nos expériences, nous calculons à chaque étape de réduction, le pourcentage de clauses supprimées, c'est-à-dire le nombre de clauses dominées (qui sont supprimées) sur le nombre total de clauses apprises à cette étape. Ceci permet d'obtenir un pourcentage moyen de clauses apprises supprimées par instance résolue. En prenant toutes les instances résolues de la SAT-RACE 2015, la moyenne des pourcentages moyens de clauses apprises supprimées est égale à 0.36 avec un écart-type de 0.16.

Les figures 3 et 4 traçent pour chaque instance résolue de la SAT-RACE 2015 (axe-X), le pourcentage moyen de clauses apprises supprimées (courbe de couleur rouge avec l'axe Y de gauche) contre respectivement le temps total de résolution et le temps moyen de résolution (courbe de couleur verte avec l'axe Y de droite). Pour chaque instance résolue, le temps moyen de résolution est obtenu en divisant son temps total de résolution par le nombre de réductions éffectuées avant de résoudre l'instance.

La courbe de couleur rouge des pourcentages moyen de clauses apprises supprimées présente une forte variation des pourcentages de réduction de 0.11 à 0.89, avec une valeur moyenne égal à 0.36 et un écart-type de 0.16. Il ressort de cette figure que le pourcentage moyen de clauses apprises supprimées est inférieur à 50% sur 200 instances parmi les 248 instances résolues. Notre stratégie actuelle qui utilise une seule clause non dominée à chaque étape de réduction est satisfaisante par rapport au temps d'exécution, même s'il est possible d'étendre cette stratégie pour une réduction avec plusieurs clauses non dominées. La courbe des pourcentages moyens de clauses apprises supprimées montre également 17 intances ayant un pourcentage moyen de clauses apprises supprimées égal à 0. Ces 17 instances correspondent aux instances résolues par le solveur sans avoir à réduire la base des clauses apprises.

La figure 3 montre d'une part, les instances dont les temps de résolution sont petits mais avec un pourcentage moyen élevé de clauses apprises supprimées, et

d'autre part, les instances dont les temps de résolution sont élevés mais avec un faible pourcentage moyen de clauses apprises supprimées. La même remarque est aussi valide avec la figure 4 où nous utilisons le temps moyen de résolution au lieu du temps total de résolution.

Ceci montre clairement que le nombre de clauses apprises supprimées à chaque étape de réduction n'est pas le seul composant des solveurs SAT qui influe sur le temps de résolution d'une instance SAT donnée. D'autres composants clés des solveurs SAT CDCL tels que les polices de redémarrage [11] et les heuristiques de selection des variables basées sur les activités [15] ont aussi une influence sur le temps de résolution.

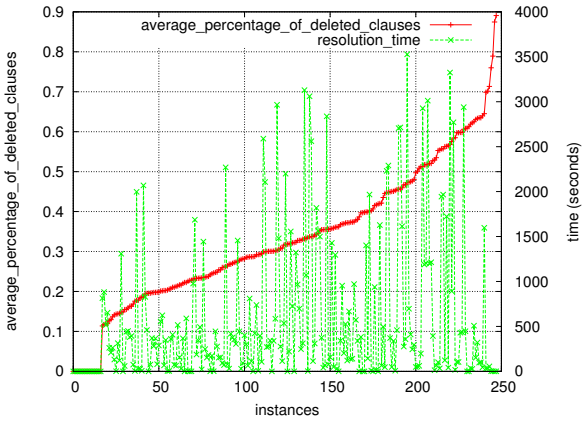


FIGURE 3 – **Pourcentage moyen de clauses apprises supprimées vs temps de résolution pour chaque instance.**

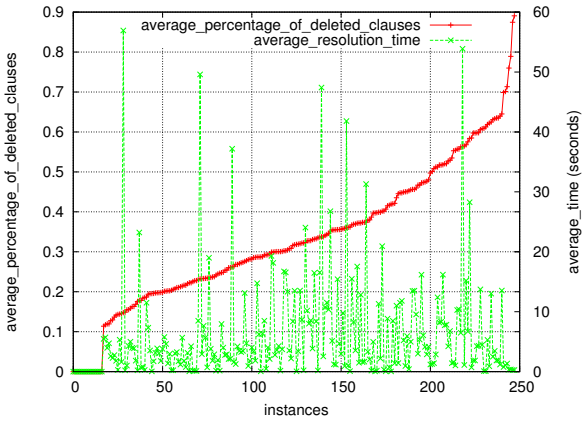


FIGURE 4 – **Pourcentage moyen de clauses apprises supprimées vs temps de résolution moyen pour chaque instance.**

5 Conclusion et Perspectives

Dans ce papier, nous proposons une approche qui traite le problème de la gestion de la base des clauses apprises. Nous avons montré que l'idée de la relation de dominance entre mesures de qualité des clauses apprises est un bon moyen pour tirer profit de chaque mesure. Cette approche n'est pas altérée par l'abondance des mesures pertinentes qui ont fait l'objet de plusieurs travaux. L'approche proposée évite un autre problème non trivial qui est celui de déterminer la quantité de clauses apprises à supprimer à chaque étape de réduction de la base des clauses apprises.

Les résultats expérimentaux montrent que l'exploitation de la relation de dominance améliore les performances des solveurs SAT CDCL, au moins sur les instances de la SAT-RACE 2015. Pour le cas de la compétition SAT 2016, nous devons encore trouver une bonne relation de dominance. Les catégories d'instances sont également un problème qui devrait être exploré.

A notre connaissance, c'est la première fois que la relation de dominance a été utilisée dans le domaine de la satisfiabilité pour améliorer les performances d'un solveur SAT CDCL. Notre approche ouvre des perspectives intéressantes. En effet, toute nouvelle mesure de pertinence de clauses apprises peut être intégrée dans la relation de dominance.

Remerciements

Les auteurs remercient le CRIL (Centre de Recherche en Informatique de Lens) pour leur avoir mis à disposition le serveur informatique pour les expérimentations. Merci également aux auteurs du solveur SAT *Glucose* pour la disponibilité du code source de leur solveur.

Les auteurs souhaitent également remercier la région Auvergne-Rhône-Alpes et l'Union européenne pour leur soutien financier par le biais du Fonds européen de développement régional (FEDER) dans le cadre du projet MobiPaleo.

Références

[1] Carlos Ansótegui, Jesús Giráldez-Cru, Jordi Levy, and Laurent Simon. Using community structure to detect relevant learnt clauses. In *SAT 2015*, pages 238–254, 2015.

[2] G. Audemard and L. Simon. Predicting learnt clauses quality in modern sat solvers. In *IJCAI'09*, pages 399–404, 2009.

[3] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Sais. On freezing and reac-

tivating learnt clauses. In *SAT'2011*, pages 188–200, 2011.

[4] A. Biere. Lingeling and friends entering the sat challenge 2012. In A. Balint, A. Belov, A. Diepold, S. Gerber, M. Jarvisalo, , and C. Sinz (editors), editors, *Proceedings of SAT Challenge 2012 : Solver and Benchmark Descriptions*, pages 33–34, University of Helsinki, 2012. vol. B-2012-2 of Department of Computer Science Series of Publications B.

[5] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 421–430, 2001.

[6] Slim Bouker, Rabie Saidi, Sadok Ben Yahia, and Engelbert Mephu Nguifo. Mining undominated association rules through interestingness measures. *International Journal on Artificial Intelligence Tools*, 23(4), 2014.

[7] Martin Davis, Gearge Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397, 1962.

[8] Niklas Eén and Armin Biere. Effective preprocessing in sat through variable and clause elimination. In *SAT'05*, pages 61–75, 2005.

[9] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *SAT 2003*, pages 502–518, 2003.

[10] Eugene Goldberg and Yakov Novikov. Berkmin : A fast and robust sat-solver. *Discrete Applied Mathematics*, 155(12) :1549 – 1561, 2007.

[11] Carla P. Gomes, Bart Selman, and Henry A. Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.

[12] Long Guo, Saïd Jabbour, Jerry Lonlac, and Lakhdar Sais. Diversification by clauses deletion strategies in portfolio parallel SAT solving. In *ICTAI 2014*, pages 701–708, 2014.

[13] Saïd Jabbour, Jerry Lonlac, Lakhdar Sais, and Yakoub Salhi. Revisiting the learned clauses database reduction strategies. *CoRR*, abs/1402.1956, 2014.

[14] Hadi Katebi, Karem A. Sakallah, and João P. Marques Silva. Empirical study of the anatomy of modern sat solvers. In *SAT 2011*, pages 343–356, 2011.

[15] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : Engineering an efficient sat solver. In *38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.

[16] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning sat solvers with restarts. In *(CP'09)*, pages 654–668, 2009.

[17] João P. Marques Silva and Karem A. Sakallah. Grasp - a new search algorithm for satisfiability. In *International Conference on Computer-Aided Design (ICCAD'96)*, pages 220–227, 1996.

[18] João P. Marques Silva and Karem A. Sakallah. GRASP : A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5) :506–521, 1999.

[19] Arnaud Soulet, Chedy Raïssi, Marc Plantevit, and Bruno Crémilleux. Mining dominant patterns in the sky. In *ICDM 2011*, pages 655–664, 2011.

[20] David Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation*, 1(1) :67–82, 1997.

[21] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design, ICCAD, San Jose, CA, USA, November 4-8*, pages 279–285, 2001.

Considération des motifs dans la détermination d'une borne supérieure de la force d'un graphe

Clément Lecat *__Corinne Lucet Chu-Min Li__

Laboratoire de Modélisation, Information et Système EA 4290
Université de Picardie Jules Verne, Amiens, France
{clement.lecat,corinne.lucet,chu-min.li}@u-picardie.fr

Résumé

Le Problème de la Somme Coloration Minimum (*MSCP*) est un problème de coloration qui associe à chaque couleur un poids. L'objectif du problème *MSCP* est de déterminer une coloration minimisant la somme des poids des couleurs. *MSCP* a des applications dans différents domaines tels que la planification ou le problème VLSI. Le nombre minimal de couleurs nécessaire à l'obtention d'une solution optimale pour *MSCP* est appelé la *force* du graphe G, notée $s(G)$. Une bonne borne supérieure de $s(G)$ permet de réduire significativement la taille de l'espace de recherche de *MSCP*. Dans ce papier, nous proposons deux nouvelles bornes supérieures, UB_A et UB_S, de $s(G)$ pour des graphes généraux. Les résultats expérimentaux réalisés sur les instances *DIMACS* et *COLOR* montrent que notre approche améliore significativement les bornes de la force pour ces instances, notamment UB_S qui améliore la borne supérieure de $s(G)$ pour 70 graphes sur 74.

Abstract

The Minimum Sum Colouring Problem (*MSCP*) is a vertex colouring problem with a weight associated to each colour. The aim of the *MSCP* is to find an optimal colouring of the vertices of a graph G with the minimum sum of the associated weights of the used colours. The *MSCP* has important applications in fields such as scheduling and VLSI design. The minimum number of colours in an optimal solution of the *MSCP* for G is called the *chromatic strength* of G and is denoted by $s(G)$. Tight upper bounds of $s(G)$ allow to significantly reduce the search space when solving the *MSCP*. In this paper, we propose two new upper bounds, UB_A and UB_S, of $s(G)$ for general graphs based on an abstraction of all possible colourings of G formulated as an ordered set of non-increasing positive integer sequences. The results of experiments on the standard benchmarks *DIMACS* and *COLOR* show that our new bounds are significantly tighter than those previously published in the literature : UB_S improves the upper bound for 70 graphs out of 74.

*Papier doctorant : Clément Lecat est auteur principal.

1 Introduction

Le problème de la Somme Coloration Minimum (*MSCP*) d'un graphe est un problème d'optimisation combinatoire *NP*-difficile [15], dont l'objectif est d'associer, en respectant la contrainte de voisinage, une couleur pondérée à chaque sommet d'un graphe G tel que la somme des poids soit minimale. La somme associée à une coloration optimale de *MSCP* est appelée la somme chromatique d'un graphe, notée $\Sigma(G)$. Le nombre minimal de couleurs nécessaire à l'obtention d'une solution optimale pour *MSCP* est appelée la force d'un graphe, notée $s(G)$.

Les différents travaux relatifs à *MSCP* peuvent être classés en deux catégories. La première catégorie regroupe l'ensemble des résultats théoriques permettant à partir de caractéristiques structurelles de certaines familles de graphes d'extraire différentes bornes de $\Sigma(G)$ ou $s(G)$ [1, 10, 20, 28]. La seconde catégorie regroupe les différentes méthodes de résolution comprenant les méthodes incomplètes telles que les métha-heuristiques, les heuristiques [2, 13, 19, 24, 27] et les méthodes complètes telles que le Branch and Bound [16]. Cependant, à notre connaissance, les meilleures techniques de résolution de *MSCP* consistent à modéliser ce dernier en un autre paradigme tel que la logique propositionelle (weighted MinSAT/MaxSAT)[16] et la programmation par contraintes [16, 22]. L'espace de résolution de *MSCP* est fortement dépendant du nombre de couleurs initiales considéré. En effet, dans l'ensemble des méthodes citées précédemment, à l'exception de weighted MinSAT/MaxSAT, la taille de l'espace de recherche de

MSCP est de $k^{|V|}$, k étant le nombre de couleurs initialement considéré. Dans le cas d'une modélisation en logique propositionnelle (weighted MinSAT/MaxSAT), la taille de ce dernier est $2^{k\times|V|}$. Par conséquent, l'élaboration d'une bonne borne supérieure pour $s(G)$ est essentielle à la résolution de *MSCP*.

Différentes bornes pour la force d'un graphe ont été établies dans la littérature pour certaines familles de graphes, telles que les arbres, les graphes d'intervalles ou les k-arbres partiels [10, 15, 23, 25]. Dans le cas général, peu de méthodes permettent d'extraire une borne pour la force. À notre connaissance, seules deux bornes ont été proposées, dans [23] et dans [7]. Dans ce papier, nous proposons deux nouvelles bornes supérieures pour $s(G)$ pour des graphes généraux, appelées UB_A et UB_S, extraites de l'exploration d'une abstraction de l'ensemble des colorations de G appelée *motifs*.

Un motif est une séquence décroissante d'entiers positifs, où chaque entier représente la cardinalité d'un sous-ensemble de sommets du graphe G partageant une même couleur. La notion de motif a été introduite dans [3, 4] pour la résolution de *MSCP* pour des graphes de type P_4-sparse et dans [17] pour borner la somme chromatique. À notre connaissance, elle n'a jamais été utilisée pour borner $s(G)$. Les résulats expérimentaux realisés sur les benchmarks *DIMACS* et *COLOR* [5, 9] montrent que UB_A et UB_S améliorent significativement la borne supérieure de $s(G)$ en comparaison des résultats montrés dans [7, 23]. De plus, pour 8 graphes, UB_S a permis de déterminer la valeur optimale de la force.

Ce papier est organisé comme suit. La section 2 définit le problème *MSCP* et présente différentes notions autour de la force. La section 3 présente l'état de l'art de $s(G)$ et propose différentes propriétés sur les motifs permettant d'introduire UB_A et UB_S. La section 4 compare UB_A et UB_S aux bornes connues de la littérature sur un ensemble d'instances. Pour terminer, la section 5 conclut ce papier.

2 Définitions et formulations

2.1 Nombre chromatique, somme coloration minimale, clique maximum et stable maximum

Soit $G = (V, E)$ un graphe non orienté, où V représente l'ensemble des sommets ($|V| = n$) et $E \subseteq V^2$ l'ensemble des arêtes de G. L'ensemble des sommets adjacents (voisins) de $v \in V$, noté $\mathcal{N}_v$, est défini comme suit : $\mathcal{N}(v) = \{u \mid (u, v) \in E\}$. Le degré $d(v)$ est le nombre de voisins de v i.e. $|\mathcal{N}(v)|$. Le degré d'un graphe G, noté $\Delta(G)$, est $\max\{d(v) \mid v \in V\}$.

Une clique C est un sous-ensemble de V tel que $\forall u, v \in C$, $(u, v) \in E$. Le problème de la clique maximum d'un graphe (MaxClique) consiste à rechercher une clique dans G dont la cardinalité est maximum.

Un stable S est un sous-ensemble de V tel que $\forall u, v \in S$, $(u, v) \notin E$. Le problème de stable maximum d'un graphe G consiste à rechercher un stable dans G dont la cardinalité est maximum.

Le graphe complémentaire de G est défini comme suit : $\overline{G} = (V, \overline{E})$, où $\overline{E} = V^2 \setminus E$. Nous remarquons qu'une clique de G est un stable dans $\overline{G}$ et vice versa.

Une coloration d'un graphe G avec k couleurs (k-coloration) est une fonction $c : V \mapsto \{1, 2, \ldots, k\}$ qui assigne à chaque sommet $v \in V$ une couleur $c(v)$. Une coloration est dite valide si $\forall (u, v) \in E$, $c(u) \neq c(v)$. Une k-coloration valide est representée comme suit : $X = \{X_1, X_2, \ldots, X_k\}$, où $X_i = \{v \in V \mid c(v) = i\}$ est une *classe couleur*. Nous remarquons qu'une classe couleur est un stable de G.

Le nombre de couleurs minimal nécessaire à l'obtention d'une coloration valide d'un graphe G est appelé le *nombre chromatique* de G, noté $\chi(G)$. La recherche du nombre chromatique d'un graphe est un problème *NP*-difficile [14].

Pour *MSCP*, un poids $w_i = i$ est associé à chaque couleur i. La somme coloration d'une coloration X, notée $\Sigma(X)$, est définie comme suit :

$$\Sigma(X) = 1 \times |X_1| + 2 \times |X_2| + ... + k \times |X_k|$$

Exemple 1 *Considérons le graphe G de la Figure 1. $X = \{\{a, e\}_1, \{b\}_2, \{c, d, f\}_3\}$ est une coloration valide de G telle que les sommets a et e sont colorés avec la couleur 1, le sommet b avec la couleur 2 et les sommets c, d et g avec la couleur 3. La somme coloration associée est $\Sigma(X) = 1 \times 2 + 2 \times 1 + 3 \times 3 = 13$.*

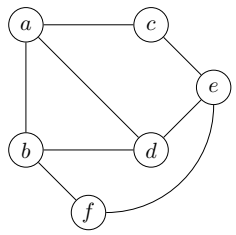


FIGURE 1 – Graphe simple.

Comme précisé dans l'introduction, l'objectif de *MSCP* est de déterminer une coloration valide X telle que $\Sigma(G) = \min\{\Sigma(X) |\ X$ *est une coloration valide de* $G\ \}$.

2.2 Coloration Majeure

Une classe couleur X_i étant un stable, il est possible de permuter deux classes couleurs X_i et X_j d'une colo-

ration X sans en affecter la validité. Ainsi, l'ensemble des permutations des classes couleurs d'une coloration X forme une classe d'équivalence notée $\Psi(X)$. Deux colorations de $\Psi(X)$ sont dites symétriques. Cependant, si l'ensemble des colorations symétriques ont un nombre de couleurs identique, il n'en est pas de même pour la somme coloration. En effet, pour le graphe G de la Figure 1, la somme coloration de $\{\{a,e\}_1,\{b\}_2,\{c,d,f\}_3\}$ est égale à 13 alors que la somme coloration de $\{\{a,e\}_1,\{c,d,f\}_2,\{b\}_3\}$ est égale à 11. L'objectif de *MSCP* est de minimiser la somme des poids, il est donc possible de ne considérer que la coloration ayant la plus petite somme coloration de $\Psi(X)$. Une telle coloration est appelée *coloration majeure* et se définit comme suit :

Définition 1 *Une coloration majeure, notée X^m, est une coloration $X^m = \{X_1, X_2, \ldots, X_k\}$ telle que $|X_1| \geq |X_2| \geq \ldots \geq |X_k|$.*

Exemple 2 *Sur la Figure 1, la coloration $X = \{\{c,d,f\}_1,\{a,e\}_2,\{b\}_3\}$ est une coloration majeure. $\Sigma(X^m) = 10$.*

La propriété suivante est une conséquence directe de la Définition 1.

Propriété 1 *Soit $\Psi(X)$ un ensemble de colorations symétriques d'une coloration X et X^m une coloration majeure de $\Psi(X)$, alors $\forall X' \in \Psi(X)$, $\Sigma(X^m) \leq \Sigma(X')$.*

Ainsi, seules les colorations majeures sont considérées dans la suite de ce papier et sont simplement notées X.

2.3 Motifs

À toute coloration majeure X de cardinalité k, il est possible d'associer une séquence décroissante p d'entiers positifs, telle que $p = (|X_1|, |X_2|, \ldots, |X_k|)$. Une telle séquence est appellée *motif* de X. Le i^{ieme} entier de p est noté $p[i]$. Sa valeur est le nombre de sommets de la classe X_i : $p[i] = |X_i|$. La longueur de p est le nombre de couleurs utilisées par X, notée $|p|$ ($|p| = k$). La somme coloration associée à un motif est définie par l'Équation 1.

$$\Sigma(X) = \Sigma(p) = 1 \times p[1] + 2 \times p[2] + ... + k \times p[k] \quad (1)$$

Exemple 3 *Le motif correspondant à $X = \{\{c,d,f\}_1,\{a,e\}_2,\{b\}_3\}$ est $p = (3,2,1)$, avec $p[1] = 3$, $p[2] = 2$ et $p[3] = 1$. $\Sigma(X) = \Sigma(p) = 10$.*

Nous remarquons qu'un motif peut être associé à plusieurs colorations majeures. En effet, le motif correspondant aux colorations majeures $\{\{c,d,f\}_1,\{b,e\}_2,\{a\}_3\}$ et $\{\{a,e,f\}_1,\{b,c\}_2,\{e\}_3\}$ de la Figure 1 est $p = (3,2,1)$. Ainsi, la considération de l'ensemble des motifs permet une réduction de l'espace de recherche de *MSCP*.

Pour un graphe $G = (V,E)$ tel que $|V| = n$, $\phi(n)$ représente l'ensemble de tous les motifs et $\phi(n,k)$ représente l'ensemble des motifs tel que $|p| = k$. Les ensembles $\phi(n,k)$ dans $\phi(n)$ sont triés par ordre croissant de k, alors que les motifs dans $\phi(n,k)$ sont triés par ordre lexicographique décroissant (Définition 2) :

Définition 2 *Soit p_k et q_k deux motifs de $\phi(n,k)$. p_k précède lexicographiquement (ou est plus grand que) q_k ssi $p_k[1] > q_k[1]$ ou $\exists t > 0$ tel que $p_k[x] = q_k[x]$ pour tout $x < t$ et $p_k[t] > q_k[t]$.*

L'ordre lexicographique décroissant permet d'assigner un index i à chaque motif de $\phi(n,k)$: p_k^i est le $i^{ème}$ motif de $\phi(n,k)$ dans l'ordre lexicographique décroissant. La Table 1 présente l'ensemble des motifs de $\phi(8)$ ainsi que leur somme coloration.

ϕ(n,k)	p$\in \phi$(n,k)	$\Sigma(p)$
ϕ(8,1)	(8)	8
ϕ(8,2)	(7,1)	9
	(6,2)	10
	(5,3)	11
	(4,4)	12
ϕ(8,3)	(6,1,1)	11
	(5,2,1)	12
	(4,3,1)	13
	(4,2,2)	14
	(3,3,2)	15
ϕ(8,4)	(5,1,1,1)	14
	(4,2,1,1)	15
	(3,3,1,1)	16
	(3,2,2,1)	17
	(2,2,2,2)	20
ϕ(8,5)	(4,1,1,1,1)	18
	(3,2,1,1,1)	19
	(2,2,2,1,1)	21
ϕ(8,6)	(3,1,1,1,1,1)	23
	(2,2,1,1,1,1)	24
ϕ(8,7)	(2,1,1,1,1,1,1)	29
ϕ(8,8)	(1,1,1,1,1,1,1,1)	36

TABLE 1 – $\phi(8)$

La représentation de l'ensemble des motifs de $\phi(n)$ est équivalente au partitionnement d'un entier. Cette dernière croît donc exponentiellement selon n. Hardy

et Ramanujan dans [8] donne une approximation du nombre de partitions existantes pour un entier n :

$$|\phi(n)| \sim \frac{1}{4n\sqrt{3}}\ e^{\pi\sqrt{2n/3}} \qquad (2)$$

Il n'est pas nécessaire d'énumérer l'ensemble des motifs mais seulement un très petit sous-ensemble dans notre approche. La sous-section suivante présente la relation de dominance entre deux motifs.

2.4 Relation de dominance

La relation de dominance, notée $\succeq$, entre les motifs de $\phi(n)$ est introduite par Bonomo et Valencia-Pabon [3, 4] pour calculer la somme chromatique pour un sous-ensemble de la famille des graphes P_4-sparses et pour extraire une borne supérieure de la somme chromatique pour l'ensemble des graphes P_4-sparses. Cependant, il est nécessaire d'adapter leur définition de dominance pour notre approche.

Définition 3 *Soit p et q deux motifs de $\phi(n)$. p domine q, noté $p \succeq q$, ssi $\forall t$ tel que $1 \leq t \leq \min\{|p|, |q|\}$, $\sum_{x=1}^{t} p[x] \geq \sum_{x=1}^{t} q[x]$.*

Exemple 4 *Soit p et q deux motifs, tels que $p = (5, 2, 1)$ et $q = (4, 3, 1)$.*
Pour $t = 1$: $5 > 4$;
Pour $t = 2$: $5 + 2 \geq 4 + 3$;
Pour $t = 3$: $5 + 2 + 1 \geq 4 + 3 + 1$.
Donc $p \succeq q$.

Nous remarquons que la relation de dominance est un ordre partiel. En effet, les motifs $(3, 1, 1, 1, 1, 1)$ et $(2, 2, 2, 2)$ ne sont pas comparables.

La relation de dominance est intéressante pour *MSCP* car elle a une conséquence directe sur la somme coloration, comme l'indique la Propriété 2.

Propriété 2 *Soit un graphe G, p et q deux motifs associés aux colorations valides X et X' de G. Si $p \succeq q$, alors $\Sigma(X) \leq \Sigma(X')$.*

3 Borne supérieure de la force d'un graphe

3.1 État de l'art

Dans la littérature, différentes relations ont été établies entre les propriétés structurelles d'un graphe G (degré, nombre chromatique) et sa somme chromatique [1, 4, 11, 26], mais peu de propriétés structurelles ont permis d'établir une borne supérieure de $s(G)$ pour un graphe général. La première est une borne supérieure triviale proposée par Mitchem et Morriss dans [23] enoncée dans la Propriété 3.

Propriété 3 *Soit un graphe G et $\Delta(G)$ son degré.*

$$s(G) \leq \Delta(G) + 1$$

La seconde, énoncée dans la Propriété 4, fut proposée par Hajiabolhassan et al [7]. Ces derniers considèrent la cardinalité d'une coloration valide afin de borner la force.

Propriété 4 *Soit G un graphe, et X une coloration avec k couleurs.*

$$s(G) \leq \left\lceil \frac{\Delta(G) + \chi(G)}{2} \right\rceil \leq \left\lceil \frac{\Delta(G) + k}{2} \right\rceil$$

D'autres travaux ont permis de borner la force pour certaines familles de graphe telles que les arbres [15, 23], les k-arbres partiels [10] ou encore les graphes d'intervalles [25]. Ces bornes sont données dans les Propriétés 5, 6, 7 et 8.

Propriété 5 *Soit T un arbre de n sommets. $s(T)$ est borné par $\Omega(log(n))$.*

Propriété 6 *Soit T un arbre et ρ la cardinalité de l'ensemble des sommets du chemin maximum de T.*

$$s(T) \leq 1 + \left\lceil \frac{\rho}{2} \right\rceil$$

Propriété 7 *Soit un k-arbre partiel G de n sommets. $s(G)$ est borné par $\Omega(k \times log(n))$.*

Propriété 8 *Soit un graphe d'intervalles G.*

$$s(G) \leq 2 \times \chi(G) - 1$$

3.2 Principe de construction de UB_A et UB_S

Soit X une coloration valide d'un graphe G et p le motif associé à X. L'objectif est de déterminer $k_t \in [1..n]$ tel que pour tout motif q de $\phi(n)$ avec $|q| \geq k_t$, $p \succeq q$ (cf. Équation 3).

$$\forall q \in \mho_{k_t} = \bigcup_{x=k_t}^{n} \phi(n, x),\ p \succeq q \qquad (3)$$

Nous remarquons que l'ensemble $\mho_{k_t}$ (Équation 3) n'est pas vide. En effet, il contient au minimum le motif trivial $(1, 1, \ldots, 1)$ de $\phi(n, n)$ pour $k_t = n$. Comme $\forall q \in \mho_{k_t}$, $p \succeq q$ d'après la Propriété 2, $\mho_{k_t}$ ne contient aucune coloration valide dont la somme coloration est inférieure à celle de p. Par conséquent, k_t est une borne supérieure de $s(G)$.

3.3 UB_A, une borne supérieure algébrique de s(G)

La construction de la borne algébrique UB_A s'appuie sur un ensemble de propriétés de la relation de dominance dans l'ensemble ordonné de motifs $\phi(n)$.

Propriété 9 *Soit $G = (V, E)$, tel que $|V| = n$, et un entier $k \leq n$. Le motif $p_k^1 = (n - k + 1, 1, 1, \ldots, 1)$ domine tout motif de $\phi(n, k)$.*

Preuve 1 *Soit $q \in \phi(n, k)$. Alors, $\forall t$ tel que $1 \leq t \leq k$, $\sum_{x=t+1}^{k} p_k^1[x] \leq \sum_{x=t+1}^{k} q[x]$ car $p_k^1[x] = 1 \leq q[x]$ quand $x > 1$. Par conséquent, $\sum_{x=1}^{t} p_k^1[x] = n - \sum_{x=t+1}^{k} p_k^1[x] \geq n - \sum_{x=t+1}^{k} q[x] = \sum_{x=1}^{t} q[x]$.□*

Propriété 10 *Soit $G = (V, E)$ avec $|V| = n$, k et k' deux entiers tels que $1 \leq k < k' \leq n$. Le motif $p_k^1 = (n - k + 1, 1, 1, \ldots, 1)$ domine $p_{k'}^1 = (n - k' + 1, 1, 1, \ldots, 1)$.*

Preuve 2 *Puisque $n - k + 1 > n - k' + 1$, $\forall t$ tel que $1 \leq t \leq k$, $\sum_{x=1}^{t} p_k^1[x] \geq \sum_{x=1}^{t} p_{k'}^1[x]$.□*

D'après la Propriété 9 et la Propriété 10, le motif p_k^1 domine tout motif lui succédant dans $\phi(n)$. Donc la relation suivante est vérifiée : $\forall q \in \phi(n)$ tel que $|q| \geq k$, $\Sigma(p_k^1) \leq \Sigma(q)$. La somme coloration associée à un motif p_k^1 est donnée par l'Équation 4.

$$\Sigma(p_k^1) = (n - k + 1) + \sum_{x=2}^{k} x = \frac{1}{2}k^2 - \frac{1}{2}k + n \quad (4)$$

Soit X une coloration valide et k_t le plus petit nombre de couleurs tel que :

$$\Sigma(p_{k_t}^1) \geq \Sigma(X) \quad (5)$$

En considérant l'Équation 4, l'Équation 5 devient

$$\frac{1}{2}k_t^2 - \frac{1}{2}k_t + n - \Sigma(X) \geq 0 \quad (6)$$

L'Équation 6 est valide ssi

$$k_t \geq \frac{1 + \sqrt{1 + 8 \times (\Sigma(X) - n)}}{2} \quad (7)$$

Notre borne algébrique UB_A est définie par l'Équation 8.

$$UB_A = \max(\left\lceil \frac{1 + \sqrt{1 + 8 \times (\Sigma(X) - n)}}{2} \right\rceil - 1, |X|) \quad (8)$$

Nous remarquons que UB_A ne considère aucune information structurelle de G. Afin d'améliorer cette borne, nous proposons dans la prochaine sous-section de considérer, en plus d'une coloration valide du graphe, la valeur de son stable maximum.

3.4 UB_S, une borne supérieure algorithmique de s(G)

Comme mentionné dans la Section 2 une clique maximum de $\overline{G}$ est un stable maximum de G. Notre nouvelle borne algorithmique, UB_S, est basée sur la cardinalité de la clique maximum de $\overline{G}$ trouvée par le solveur IncMaxCLQ [18]. La construction de la borne supérieure de la force UB_S, s'appuie sur la Propriété 11.

Propriété 11 *Soit $\alpha(G)$ la cardinalité du stable maximum de G. Tout motif p, tel que $p[1] > \alpha(G)$, ne peut correspondre à une coloration valide de G.*

En conséquence, tout motif p de $\phi(n)$ tel que $p[1] > \alpha(G)$ peut être exclu de l'espace de recherche. De plus, nous remarquons qu'étant donné deux classes couleurs X_i et X_j d'une coloration X, tel que $i < j$, il est parfois possible de transférer un sommet x de X_j à X_i afin d'obtenir une nouvelle coloration X'. Clairement, $\Sigma(X') < \Sigma(X)$. Nous appelons cette transformation de p, *left-shifting*. Cependant, une opération de left-shifting n'est pas toujours réalisable. En effet, il est nécessaire que la résultante d'une telle opération reste un motif, c'est-à-dire, une suite d'entiers décroissante, tel que $\forall x < |p|$, $p[x] > 0$. Ainsi, un motif de $\phi(n, k)$ qui ne peut être transformé en un autre motif de $\phi(n, k)$ par une opération de left-shifting sans incrémenter le premier entier est appelé *motif majeur*.

Intuitivement, un motif majeur de $\phi(n, k)$ contient le nombre maximum (β) d'entiers égal à son premier entier, noté λ ($\left\lceil \frac{n}{k} \right\rceil \leq \lambda \leq n - k + 1$). Le nombre $n - \beta \times \lambda$ de sommets restant doit être partitionné en $k - \beta$ entiers positifs. Donc, β est l'entier maximum satisfaisant $n - \beta \times \lambda \geq k - \beta$, ou $\beta \leq \frac{n-k}{\lambda-1}$ après exclusion du motif trivial (1, 1, ..., 1) et en supposant que $\lambda > 1$. Donc, $\beta = \left\lfloor \frac{n-k}{\lambda-1} \right\rfloor$.

Nous définissons formellement un motif majeur dans la Définition 4.

Définition 4 *Soit λ et β deux entiers tels que $\left\lceil \frac{n}{k} \right\rceil \leq \lambda \leq n - k + 1$ et $\beta = \left\lfloor \frac{n-k}{\lambda-1} \right\rfloor$. Un motif majeur de $\phi(n, k)$ est un motif p_k^i avec les propriétés structurelles suivantes :*

1. *$p_k^i[x] = \lambda$, si $1 \leq x \leq \beta$;*
2. *$p_k^i[x] = n - \beta \times \lambda - (k - \beta - 1)$, si $x = \beta + 1$;*
3. *$p_k^i[x] = 1$, si $\beta + 1 < x \leq k$.*

Exemple 5 *Si nous nous rapportons à la Table 1, les motifs majeurs de* $\phi(8,4)$ *sont* $(5,1,1,1)$, $(4,2,1,1)$, $(3,3,1,1)$, $(2,2,2,2)$. *Pour le motif* $(4,2,1,1)$, $\lambda = 4$, $\beta = 1$ *et pour le motif* $(3,3,1,1)$, $\lambda = 3$, $\beta = 2$.

Les propriétés suivantes sont des conséquences de la structure des motifs majeurs.

Propriété 12 *Si* p *et* q *sont deux motifs de* $\phi(n,k)$ *tels que* p *est majeur et* $p[1] = q[1]$ *alors* $p \succeq q$.

Preuve 3 *Soit* $\beta = \left\lfloor \frac{n-k}{p[1]-1} \right\rfloor$. *Puisque* p *est majeur, nous avons* $p[1] = p[2] = \ldots = p[\beta] = q[1] \geq q[2] \geq \ldots \geq q[\beta]$. *Donc,* $\forall t$ *tel que* $1 \leq t \leq \beta$, $\sum_{x=1}^{t} p[x] \geq \sum_{x=1}^{t} q[x]$.

De plus, $\forall t$ *tel que* $\beta + 1 \leq t < k$, *nous avons* $p[t+1] = p[t+2] = \ldots = p[k] = 1 \leq q[k] \leq q[k-1] \leq \ldots \leq q[t+1]$. *Donc,* $\sum_{x=1}^{t} p[x] = n - \sum_{x=t+1}^{k} p[x] \geq n - \sum_{x=t+1}^{k} q[x] = \sum_{x=1}^{t} q[x]$.

Donc, $p \succeq q$.

Propriété 13 *Soit* p *et* q *deux motifs majeurs de* $\phi(n,k)$. *Si* $p[1] > q[1]$, *alors* $p \succeq q$.

Preuve 4 *Soit* $\beta_p = \left\lfloor \frac{n-k}{p[1]-1} \right\rfloor$ *et* $\beta_q = \left\lfloor \frac{n-k}{q[1]-1} \right\rfloor$. *Il est clair que* $\beta_p < \beta_q$. *Nous avons* $p[1] = p[2] = \ldots = p[\beta_p] > q[1] = q[2] = \ldots = q[\beta_p]$. *Par conséquent,* $\forall t$ *tel que* $1 \leq t \leq \beta_p$, $\sum_{x=1}^{t} p[x] > \sum_{x=1}^{t} q[x]$.

De plus, $\forall t$ *tel que* $\beta_p + 1 \leq t \leq k$, *nous avons* $p[t+1] = p[t+2] = \ldots = p[k] = 1 \leq q[k] \leq q[k-1] \ldots, \leq q[t+1]$, *implique que la* $\sum_{x=1}^{t} p[x] = n - \sum_{x=t+1}^{k} p[x] \geq n - \sum_{x=t+1}^{k} q[x] = \sum_{x=1}^{t} q[x]$. *Donc,* $p \succeq q$. □

Les motifs de $\phi(n,k)$ sont triés dans l'ordre lexicographique décroissant. Donc, la Propriété 12 et la Propriété 13 ont pour conséquence qu'un motif majeur de $\phi(n,k)$ domine tout motif lui succédant dans $\phi(n,k)$ (Propriété 14). La somme coloration associée à un motif majeur p_k^i, tel que $p_k^i[1] = \alpha(G)$, est donc minimale dans $\phi(n,k)$ pour G.

Propriété 14 *Soit* $p_k^i \in \phi(n,k)$ *un motif majeur et* $p_k^j \in \phi(n,k)$ *un motif tel que* $j > i$. *Alors,* $p_k^i \succeq p_k^j$.

La Propriété 15 est fondamentale pour la construction de UB_S. En effet, cette dernière introduit la relation de dominance entre un motif majeur et tout motif de $\phi(n)$.

Propriété 15 *Soit* k *et* k' *deux entiers tels que* $k < k'$ *et* $\mho_{k'} = \bigcup_{y=k'}^{n} \{\phi(n,y)\}$. *Si* p_k^i *est un motif majeur alors* $\forall q \in \mho_{k'}$ *tel que* $q[1] \leq p_k^i[1]$, $p_k^i \succeq q$.

Preuve 5 *Soit* $\beta = \left\lfloor \frac{n-k}{p_k^i[1]-1} \right\rfloor$. *Puisque* p_k^i *est majeur, nous avons* $p_k^i[1] = p_k^i[2] = \ldots = p_k^i[\beta] \geq q[1] \geq q[2] \geq \ldots \geq q[\beta]$. *Par conséquent,* $\forall t$ *tel que* $1 \leq t \leq \beta$, $\sum_{x=1}^{t} p_k^i[x] \geq \sum_{x=1}^{t} q[x]$.

De plus, $\forall t$ *tel que* $\beta + 1 \leq t \leq k$, *nous avons* $p_k^i[t+1] = p_k^i[t+2] = \ldots = p_k^i[k] = 1 \leq q[k] \leq q[k-1] \leq \ldots \leq q[t+1]$ *(*q *est une séquence d'entier positif décroissante), implique* $\sum_{x=1}^{t} p_k^i[x] = n - \sum_{x=t+1}^{k} p_k^i[x] \geq n - \sum_{x=t+1}^{k} q[x] = \sum_{x=1}^{t} q[x]$. *Donc,* $p_k^i \succeq q$. □

Étant donnée une coloration X d'un graphe G, il suffit de déterminer le plus petit k pour lequel il existe un motif majeur p_k^i, avec $p_k^i[1] = \alpha(G)$, tel que $\Sigma(X) < \Sigma(p_k^i)$. En effet, d'après la Propriété 15, $\forall q \in \phi(n)$, tel que $q[1] < p_k^i[1]$ et $|q| \geq k$, $p_k^i \succeq q$. Ainsi, rien ne sert de considérer plus de $k-1$ couleurs, aucune coloration avec une somme inférieure ne pourra être trouvée dans cette partie de l'ensemble des solutions.

L'Algorithme 1, $UB_S(n, \alpha, X)$, calcule suivant ce principe une borne supérieure de la force d'un graphe de n sommets, dont le stable maximum (ou une borne supérieure) est α^* et pour lequel nous disposons d'une coloration valide X. La fonction $constructMajorMotif(\lambda, k)$ construit un motif majeur $p \in \phi(n,k)$, tel que $p[1] = \lambda$, comme précisé dans la Définition 4. La fonction $computeSumColoring(p)$ calcule la somme coloration associée au motif p (Équation 1). La complexité de ces deux fonctions est en $O(k)$. Puisque $k \leq n$, la complexité de l'Algorithme 1 est donc en $O(n^2)$.

Nous notons que si l'extraction d'une clique maximum d'un graphe G est NP-difficile, il est cependant plus facile de résoudre le problème $MaxClique$ que de déterminer le nombre chromatique ou la somme chromatique d'un graphe G. En effet, l'extraction d'une clique maximum d'un graphe aléatoire de densité $0,5$ tel que $|V| = 200$ ne prend que quelques secondes. Cependant, aucun algorithme exact n'est capable de trouver le nombre chromatique ou la somme chromatique pour un tel graphe en un temps raisonnable.

4 Résultats expérimentaux

Dans cette section, nous comparons notre borne supérieure algébrique (UB_A) et algorithmique (UB_S) de

Algorithme 1 : $UB_S(n, \alpha^*, X)$

Entrées : n le nombre de sommets d'un graphe G, α^* la cardinalité (ou une borne supérieure) du stable maximum de G et X une coloration valide de G

Sorties : Une borne supérieure de $s(G)$

```
début
    SUM ← 0 ;
    k ← |X| ;        /*initialise k */
    λ ← α*;
    tant que SUM ≤ Σ(X) faire
        k ← k + 1 ;
        si λ > n − k + 1 alors
            λ ← n − k + 1;
        p ← constructMajorMotif(λ, k);
        SUM ← computeSumColoring(p);
    retourner k − 1;
```

la force d'un graphe avec la borne proposée par Hajiabolhassan et al. [7] (UB_{hmt}). Pour déterminer la cardinalité d'un stable maximum de G, nous cherchons la clique maximum du graphe complémentaire $\overline{G}$ à l'aide du solveur IncMaxCLQ [18]. Bien que le problème de stable maximum soit *NP*-difficile, nous remarquons que pour un grand nombre de graphes, la solution optimale est obtenue instantanément. Quand il n'est pas possible de déterminer la solution optimale en un temps raisonable, une borne supérieure de $\alpha(G)$ est utilisée. Nous avons mené nos expérimentations sur un processeur Intel Westmere Xeon E7-8837 de 2.66GHz sur les célèbres benchmarks *COLOR* [9] et *DIMACS* [5].

Le tableau 2 montre les résultats expérimentaux. Pour chaque graphe G, sont représentés dans les différentes colonnes, le nombre de sommets $|V|$, la densité du graphe d, la meilleure borne connue du nombre chromatique k^* [6, 13, 21] (entre parenthèses quand k^* n'est pas optimale), la meilleure borne connue de la somme chromatique Σ^* [12, 13], le degré du graphe $\Delta(G)$, la meilleure borne supérieure connue pour la cardinalité du stable α^* de G (entre parenthèses quand α^* n'est pas optimale). Time_s est le temps en secondes pour déterminer α^*, UB_{hmt} le résultat de la borne de Hajiabolhassan et al. (Propriété 4), UB_A le résultat de notre borne algébrique et UB_S le résultat de notre borne algorithmique.

Les résultats du tableau 2 montrent que notre simple borne algébrique UB_A permet d'améliorer la borne supérieure de la force pour 43 instances sur 74. Pour notre borne UB_S, nous améliorons considérablement la borne supérieure de la force pour 70 instances sur 74. Pour les graphes de la famille *inithx*, les résultats de UB_{hmt} sont respectivement 278, 286 et 287, pour UB_S 72, 48 et 48 (UB_A 75, 54 et 53). Nous pouvons donc considérer 200 couleurs de moins pour la résolution de *MSCP* pour ces graphes. De plus, pour 8 instances (le450_5c, le450_5d, myciel3, queen5_5, queen7_7, queen8_12, flat300-20-0 et flat1000-50-0) $UB_S = k^* = \chi(G)$ ce qui permet de prouver l'optimalité de la force. UB_{hmt} donne de meilleurs résultats que UB_S pour les graphes *DSJC500.1*, *DSJC1000.1*, *games120*, *miles250*. Une des raisons est que la qualité de notre borne supérieure de $\alpha(G)$ est trop faible sur certaines instances. Mais pour la quasi totalité des instances, ces résultats montrent l'efficacité de notre borne UB_S dans la réduction du nombre de couleurs à considérer lors de la résolution de *MSCP*.

5 Conclusion

Dans ce papier, nous nous sommes intéressés à la force $s(G)$ d'un graphe général G, définie comme le nombre minimal de couleurs nécessaire pour atteindre une solution optimale de *MSCP*. Nous avons proposé deux nouvelles bornes supérieures de $s(G)$, UB_A et UB_S, basées sur une coloration X valide de G et la notion de motifs. Nous avons premièrement établi un ordre total parmi tous les motifs de G, puis construit UB_A et UB_S en identifiant un motif dont la somme coloration est supérieure à celle de X qui domine les motifs lui succédant.

Contrairement à UB_A qui n'exploite aucune propriété structurelle de G, UB_S considère la cardinalité du stable maximum de G combinée à la notion de *motif majeur*. Les résultats ont montré que sur les instances *DIMACS* et *COLOR*, UB_A et UB_S permettent d'améliorer (de manière importante pour UB_S) la borne supérieure de la force face à celle proposée par Hajiabolhassan et al.

Une des perspectives de nos travaux est d'intégrer plus de propriétés structurelles afin d'améliorer UB_S et de développer des algorithmes efficaces pour la résolution de *MSCP*.

Références

[1] Amotz Bar-Noy and Guy Kortsarz. Minimum color sum of bipartite graphs. *Journal of Algorithms*, 28(2) :339–365, 1998.

[2] Una Benlic and Jin-Kao Hao. A study of breakout local search for the minimum sum coloring problem. In *Simulated Evolution and Learning*, pages 128–137. Springer, 2012.

[3] Flavia Bonomo and Mario Valencia-Pabon. Minimum sum coloring of P_4-sparse graphs. *Electro-*

nic Notes in Discrete Mathematics, 35 :293–298, 2009.

[4] Flavia Bonomo and Mario Valencia-Pabon. On the minimum sum coloring of P_4-sparse graphs. *Graphs and Combinatorics*, 30(2) :303–314, 2014.

[5] ftp ://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/.

[6] Stefano Gualandi and Federico Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1) :81–100, 2012.

[7] Hossein Hajiabolhassan, Mojtaba L Mehrabadi, and Ruzbeh Tusserkani. Minimal coloring and strength of graphs. *Discrete Mathematics*, 215(1) :265–270, 2000.

[8] Godfrey H Hardy and Srinivasa Ramanujan. Asymptotic formulæ in combinatory analysis. *Proceedings of the London Mathematical Society*, 2(1) :75–115, 1918.

[9] http ://mat.gsia.cmu.edu/COLOR/instances.html.

[10] Klaus Jansen. The optimum cost chromatic partition problem. In *Algorithms and complexity*, pages 25–36. Springer, 1997.

[11] Klaus Jansen. Approximation results for the optimum cost chromatic partition problem. *Journal of Algorithms*, 34(1) :54–89, 2000.

[12] Yan Jin and Jin-Kao Hao. Hybrid evolutionary search for the minimum sum coloring problem of graphs. *Information Sciences*, 352 :15–34, 2016.

[13] Yan Jin, Jin-Kao Hao, and Jean-Philippe Hamiez. A memetic algorithm for the minimum sum coloring problem. *Computers & Operations Research*, 43 :318–327, 2014.

[14] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[15] Ewa Kubicka and Allen J Schwenk. An introduction to chromatic sums. In *Proceedings of the 17th conference on ACM Annual Computer Science Conference*, pages 39–45. ACM, 1989.

[16] Clément Lecat, Chu-Min Li, Corinne Lucet, and Yu Li. Exact methods for the minimum sum coloring problem. In *Doctoral Program of Constraint Programming (CP-DP'15)*, pages 61–69, 2015.

[17] Clément Lecat, Corinne Lucet, and Chu-Min Li. New lower bound for the minimum sum coloring problem. In *AAAI Conference on Artificial Intelligence*. AAAI, 2017.

[18] Chu-Min Li, Zhiwen Fang, and Ke Xu. Combining maxsat reasoning and incremental upper bound for the maximum clique problem. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 939–946. IEEE, 2013.

[19] Yu Li, Corinne Lucet, Aziz Moukrim, and Kaoutar Sghiouer. Greedy algorithms for the minimum sum coloring problem. In *Logistique et transports*, pages LT–027, 2009.

[20] Michal Malafiejski. *Sum coloring of graphs*, chapter 4, pages 55–66. Graph Coloring. 2004.

[21] Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2) :174–190, 2011.

[22] Maël Minot, Samba Ndojh Ndiaye, and Christine Solnon. Using cp and ilp with tree decomposition to solve the sum colouring problem. In *Doctoral program of CP 2016*, 2016.

[23] John Mitchem and Patrick Morriss. On the cost-chromatic number of graphs. *Discrete Mathematics*, 171(1) :201–211, 1997.

[24] Aziz Moukrim, Kaoutar Sghiouer, Corinne Lucet, and Yu Li. Lower bounds for the minimal sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36 :663–670, 2010.

[25] Sara Nicoloso. Sum coloring and interval graphs : a tight upper bound for the minimum number of colors. *Discrete Mathematics*, 280(1) :251–257, 2004.

[26] Mohammad R Salavatipour. On sum coloring of graphs. *Discrete Applied Mathematics*, 127(3) :477–488, 2003.

[27] Kaoutar Sghiouer, Li Yu, Corinne Lucet, and Aziz Moukrim. A memetic algorithm for the minimum sum coloring problem. In *Conférence Internationale en Recherche Opérationnelle (CIRO'2010)*, 2010.

[28] Carsten Thomassen, Paul Erdös, Yousef Alavi, Paresh J Malde, and Allen J Schwenk. Tight bounds on the chromatic sum of a connected graph. *Journal of Graph Theory*, 13(3) :353–357, 1989.

Graph	$\|V\|$	d	k^*	Σ^*	$\Delta(G)$	α^*	Time$_s$	UB_{hmt}	UB_A	UB_S
DSJC125.1	125	0.094	5	326	23	34	0	14	21	**11**
DSJC125.5	125	0.502	17	1012	75	10	0	46	43	**29**
DSJC125.9	125	0.898	44	2503	120	4	0	82	69	**58**
DSJC250.1	250	0.103	8	970	38	44	117	23	38	**21**
DSJC250.5	250	0.503	(28)	3210	147	12	0	88	77	**50**
DSJC250.9	250	0.896	72	8277	234	5	24	153	127	**105**
DSJC500.1	500	0.099	(12)	2836	68	(85)	1039	**40**	69	52
DSJC500.5	500	0.501	(48)	10886	286	13	16	167	145	**81**
DSJC500.9	500	0.901	(126)	29862	471	5	84	299	243	**185**
DSJC1000.1	1000	0.099	(20)	8991	127	(175)	1540	**74**	127	111
DSJC1000.5	1000	0.500	(83)	37575	551	15	408	317	271	**150**
DSJC1000.9	1000	0.899	(222)	103445	924	6	223	573	453	**347**
le450_5a	450	0.056	5	1350	42	(94)	2430	24	43	**13**
le450_5b	450	0.056	5	1350	42	(96)	1083	24	43	**15**
le450_5c	450	0.097	5	1350	66	90	2	36	43	**5**
le450_5d	450	0.096	5	1350	68	90	2	37	43	**5**
le450_15a	450	0.080	15	2632	99	75	45	57	67	**51**
le450_15b	450	0.080	15	2632	94	78	7	55	67	**52**
le450_15c	450	0.165	15	3487	139	41	1602	77	78	**50**
le450_15d	450	0.165	15	3505	138	41	2266	77	79	**50**
le450_25a	450	0.081	25	3153	128	91	0	77	74	**64**
le450_25b	450	0.081	25	3365	111	78	0	68	77	**66**
le450_25c	450	0.171	25	4515	179	47	24	102	91	**74**
le450_25d	450	0.172	25	4544	157	43	82	91	91	**72**
queen5_5	25	0.533	5	75	16	5	0	11	11	**5**
queen6_6	36	0.460	7	138	19	6	0	13	15	**10**
queen7_7	49	0.404	7	196	24	7	0	16	18	**7**
queen8_8	64	0.361	9	291	27	8	0	18	22	**10**
queen8_12	96	0.300	12	624	32	8	0	22	33	**12**
queen9_9	81	0.325	10	409	32	9	0	21	26	**11**
queen10_10	100	0.296	11	553	35	10	0	23	31	**12**
queen11_11	121	0.272	11	733	40	11	0	26	35	**14**
queen12_12	144	0.252	12	943	43	12	0	28	40	**15**
queen13_13	169	0.234	13	1191	48	13	0	31	46	**16**
queen14_14	196	0.219	14	1482	51	14	0	33	51	**18**
queen15_15	225	0.205	15	1814	56	15	0	36	57	**19**
queen16_16	256	0.193	16	2193	59	16	0	38	63	**21**
myciel3	11	0.363	4	21	5	5	0	5	5	**4**
myciel4	23	0.280	5	45	11	11	0	8	7	**6**
myciel5	47	0.218	6	93	23	23	0	15	10	**8**
myciel6	95	0.169	7	189	47	47	0	27	14	**11**
myciel7	191	0.130	8	381	95	95	0	52	20	**15**
fpsol2_i_1	496	0.094	65	3403	252	307	0	159	77	**75**
fpsol2_i_2	451	0.085	30	1668	346	261	0	188	50	**46**
fpsol2_i_3	425	0.096	30	1636	346	238	0	188	50	**46**
inithx.i.1	864	0.050	54	3676	502	566	0	278	75	**72**
inithx.i.2	645	0.067	31	2050	541	365	0	286	54	**48**
inithx.i.3	621	0.072	31	1986	542	360	0	287	53	**48**
mulsol.i.1	197	0.203	49	1957	121	100	0	85	60	**59**
mulsol.i.2	188	0.221	31	1191	156	90	0	94	45	**44**

Continue sur la page suivante...

Graph	$\|V\|$	d	k^*	Σ^*	$\Delta(G)$	α^*	Time$_s$	UB_{hmt}	UB_A	UB_S
mulsol.i.3	184	0.232	31	1187	157	86	0	94	45	**44**
mulsol.i.4	185	0.231	31	1189	158	86	0	95	45	**44**
mulsol.i.5	186	0.230	31	1160	159	88	0	95	45	**43**
school1	385	0.258	14	2674	282	41	2	148	68	**45**
school1-nsh	352	0.236	14	2392	232	39	0	123	64	**43**
zeroin.i.1	211	0.185	49	1822	111	120	0	80	57	**56**
zeroin.i.2	211	0.159	30	1004	140	127	0	85	40	**39**
zeroin.i.3	206	0.167	30	998	140	123	0	85	40	**39**
anna	138	0.052	11	276	71	80	0	41	17	**14**
david	87	0.108	11	237	82	36	0	47	18	**15**
huck	74	0.111	11	243	53	27	0	32	19	**16**
jean	80	0.080	10	217	36	38	0	23	17	**15**
games120	120	0.089	9	443	13	22	0	**11**	26	15
flat300-20-0	300	0.476	20	3150	160	15	0	90	76	**20**
flat300-26-0	300	0.482	26	3966	158	12	0	92	86	**36**
flat300-28-0	300	0.483	28	4238	162	12	0	95	89	**49**
flat1000-50-0	1000	0.490	50	25500	520	20	263	285	222	**50**
flat1000-60-0	1000	0.492	60	30100	524	17	296	292	242	**77**
flat1000-76-0	1000	0.493	76	37164	532	15	466	304	269	**145**
miles250	128	0.047	8	325	16	44	0	**12**	20	14
miles500	128	0.143	20	705	38	18	0	29	34	**25**
miles750	128	0.259	31	1173	64	12	0	48	46	**38**
miles1000	128	0.395	42	1666	86	8	0	64	56	**47**
miles1500	128	0.639	73	3354	106	5	0	90	81	**77**

TABLE 2: Comparaison de notre borne algébrique (UB_A) et notre borne algorithmique (UB_S) face à la borne de Hajiabolhassan et al. [7] (UB_{hmt})

Vers une exploitation dynamique de la décomposition pour les CSPs pondérés

Philippe Jégou **Hanan Kanso** **Cyril Terrioux**
Aix Marseille Univ, Université de Toulon, CNRS, ENSAM, LSIS, Marseille, France
{philippe.jegou, hanan.kanso, cyril.terrioux}@lsis.org

Résumé

Pour résoudre les problèmes de satisfaction de contraintes pondérés, les méthodes basées sur une décomposition arborescente constituent une approche intéressante selon la nature des instances considérées. Souvent, les décompositions exploitées visent à réduire la taille maximale des clusters, connue comme étant la largeur de la décomposition. En effet, l'intérêt de ce paramètre est lié à son importance par rapport à la complexité théorique de telles méthodes. À ce niveau, Min-Fill constitue l'heuristique de référence pour le calcul de décompositions. Cependant, son intérêt pratique pour la résolution de problèmes demeure limité au vu de ses multiples défauts, notamment au niveau de la restriction de la liberté de l'heuristique de choix de variables.

Ainsi, nous proposons, dans un premier temps, d'exploiter de nouvelles décompositions pour le problème d'optimisation sous contraintes. Le but de ces décompositions est de capturer des critères permettant d'augmenter l'efficacité de la résolution. Dans un second temps, nous proposons d'exploiter ces décompositions plus dynamiquement dans le sens où la résolution d'un sous-problème ne se baserait sur la décomposition que lorsque cela semble utile. Les expérimentations réalisées montrent l'intérêt pratique des nouvelles décompositions ainsi que l'apport de leur exploitation dynamique.

Abstract

When solving weighted constraint satisfaction problems, methods based on a tree-decomposition constitute an interesting approach depending on the nature of the considered instances. The exploited decompositions often aim to reduce the maximal size of the clusters, which is known as the width of the decomposition. Indeed, the interest of this parameter is related to its importance with respect to the theoretical complexity of these methods. However, its practical interest for the solving of instances remains limited if we consider its multiple drawbacks, notably due to the restriction imposed on the freedom of the variable ordering heuristic.

So, we first propose to exploit new decompositions for solving the constraint optimization problem. These decompositions aim to take into account criteria allowing to increase the solving efficiency. Secondly, we propose to use these decompositions in a more dynamic manner in the sense that the solving of a sub-problem would be based on the decomposition only when it seems useful. The performed experiments show the practical interest of these new decompositions and the benefit of their dynamic exploitation.

1 Préliminaires

Le problème de satisfaction de contraintes pondéré (Weighted Constraint Satisfaction Problem (WCSP)) offre un cadre général permettant de modéliser et de résoudre efficacement toute sorte de problèmes issus du monde réel comme par exemple, les problèmes d'allocation de fréquence [4] ou certains problèmes de bio-informatique [22]. Notre objectif, dans ce papier, est d'améliorer l'efficacité des algorithmes résolvant ce problème grâce à des approches structurelles. Pour commencer, nous rappelons la définition d'une instance WCSP :

Définition 1 *Une* instance WCSP *est définie par un triplet* (X, D, W) *avec :*

- *un ensemble* $X = \{x_1, ..., x_n\}$ *de* n variables,
- *un ensemble* $D = \{D_{x_1}, ..., D_{x_n}\}$ *de* domaines *finis de* valeurs, *à raison d'un domaine par variable,*
- *un ensemble* W *de* fonctions de coût *de taille* e. *Une fonction de coût portant sur les variables* $\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ *est une fonction qui associe à chaque tuple de* $D_{x_{i_1}} \times D_{x_{i_2}} \times \ldots \times D_{x_{i_k}}$ *un entier compris entre 0 et* k *avec* k *le coût maximum associé à un tuple complètement interdit (c'est-à-dire qui exprime une contrainte dure).*

Afin de simplifier le propos, dans la suite de la présentation, nous ne considérerons, que le cas des instances binaires, c'est-à dire des instances dont chaque fonction de coût porte sur au plus deux variables. Toutefois, ce travail peut s'étendre sans problème au cas des instances non binaires. Dans la partie expérimentale, nous exploiterons d'ailleurs indifféremment des instances binaires et des instances non binaires. Par la suite, nous noterons w_{ij} la fonction de coût binaire portant sur les variables x_i et x_j, w_i la fonction de coût unaire portant sur la variable x_i et $w_\emptyset$, la fonction de coût d'arité nulle qui correspond à un coût que doivent payer toutes les affectations.

Le coût d'une affectation complète $(v_1, \ldots, v_n) \in D_{x_1} \times \ldots \times D_{x_n}$ est définie par $w_\emptyset + \sum_{x_i \in X} w_i(v_i) + \sum_{w_{ij} \in W} w_{ij}(v_i, v_j)$. Étant donnée une instance, le problème de satisfaction de contraintes pondéré consiste à trouver le coût minimum associé à une affectation complète. Ce problème d'optimisation est bien connu pour être NP-difficile.

La résolution des instances s'effectue généralement par le biais d'algorithmes de type *séparation et évaluation* (Branch and Bound). Ces algorithmes exploitent traditionnellement deux bornes notées, souvent *clb* et *cub*, qui représentent respectivement un minorant et un majorant du coût optimum. Ils procèdent en étendant progressivement une affectation tant que le minorant courant *clb* est inférieur au majorant courant *cub*. Ce faisant, l'obtention d'une affectation complète permet de mettre à jour le majorant courant avec la valeur de la dite affectation. Ce majorant peut être initialisé avec la valeur k ou grâce à une méthode incomplète. L'efficacité d'une telle approche dépend grandement de la qualité du minorant utilisé à chaque nœud de la recherche. Sur ce point, de nombreux progrès ont été accomplis, ces dernières années, notamment grâce à la définition de différentes formes de cohérences locales [6, 9, 8, 5].

La plupart des travaux effectués dans le cadre du problème WCSP reposent sur des algorithmes de type séparation et évaluation basés sur un parcours en profondeur de l'espace de recherche. Très récemment, une approche hybridant un parcours le meilleur d'abord et un parcours en profondeur a été proposée dans [1]. Elle possède notamment l'avantage de pouvoir fournir, à tout instant, un minorant global ainsi qu'un majorant du coût optimum. Son efficacité pratique est telle qu'elle surclasse, en général, les approches existantes.

Une autre alternative consiste à exploiter la structure de l'instance. Cette structure peut être représentée par un graphe, appelé *graphe de contraintes*, dont les sommets correspondent aux variables et dont les arêtes lient deux sommets dont les variables correspondantes participent ensemble à une fonction de coût

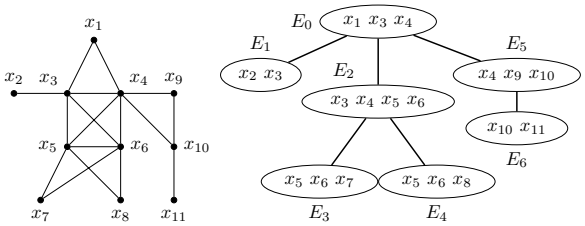


FIGURE 1 – Un graphe (a) et une de ses décompositions arborescentes optimales (b).

binaire de W. Pour exploiter cette structure, certaines méthodes [16, 23, 11, 1] reposent sur la notion de *décomposition arborescente de graphes* [20] pour identifier des sous-problèmes indépendants.

Définition 2 *Une* décomposition arborescente *d'un graphe $G = (X, C)$ est un couple (E, T) où $T = (I, F)$ est un arbre (I est un ensemble de nœuds et F un ensemble d'arêtes) et $E = \{E_i : i \in I\}$ une famille de sous-ensembles de X, telle que chaque sous-ensemble (appelé 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 le chemin entre i et j dans T, alors $E_i \cap E_j \subseteq E_k$. La largeur d'une décomposition arborescente est égale à $max_{i \in I}|E_i|-1$. La largeur arborescente dite* tree-width *w de G est la largeur minimale pour toutes les décompositions arborescentes de G.*

La figure 1 présente un graphe et une de ses décompositions arborescentes dont la largeur est 3.

L'utilisation d'une décomposition arborescente permet alors de décomposer le problème original en plusieurs sous-problèmes, d'exploiter des bornes locales de meilleure qualité et d'éviter certaines redondances dans le calcul grâce à l'enregistrement de certaines informations. D'un point de vue théorique, l'intérêt d'une telle approche réside dans sa complexité en temps qui est généralement de l'ordre de $O(n.e.d^{w^+ + 1})$ (avec w^+ la largeur de la décomposition arborescente exploitée) tandis que les méthodes classiques ont une complexité en $O(n.e.d^n)$. D'un point de vue pratique, l'exploitation simultanée des décompositions arborescentes et des cohérences locales a conduit à définir des méthodes ayant une efficacité remarquable, dépassant celles des méthodes classiques [11, 1]. Cette efficacité dépend, en partie, des décompositions employées.

Le plan de cet article est le suivant. Dans la section suivante, nous abordons les méthodes de calcul de ces décompositions. Puis, dans la section 3, nous décrivons comment exploiter dynamiquement une dé-

composition dans le cadre de la résolution d'instances WCSP. Enfin, dans la section 4, nous évaluons l'intérêt pratique de certaines décompositions ainsi que l'apport des décompositions dynamiques avant de conclure dans la section 5.

2 Calcul de décompositions arborescentes

Calculer une décomposition optimale (c'est-à-dire de largeur minimum) est un problème NP-difficile [2]. Aussi, le calcul d'une décomposition arborescente est souvent réalisé par le biais de méthodes heuristiques. Dans ce contexte, la méthode Min-Fill [21] fait office de référence. Grâce à elle, les décompositions arborescentes ont déjà été exploitées avec succès pour résoudre des instances WCSP [1]. Pour autant, leur potentiel n'a pas été pleinement exploité notamment du fait de la qualité des décompositions calculées. En effet, Min-Fill, par exemple, a été conçue pour calculer une décomposition de largeur la plus petite possible, et rien ne garantit que la décomposition produite soit adaptée du point de vue de la résolution d'instances WCSP. D'ailleurs, très récemment, il a été montré, dans [15, 12], que les décompositions produites par Min-Fill n'étaient pas dépourvues de défauts du point de vue de la résolution d'instances du problème de décision CSP et que l'emploi de décompositions spécifiques conduisaient souvent à résoudre plus efficacement les instances.

Dans [12], un cadre générique, appelé *H-TD-WT* (pour *Heuristic Tree-Decomposition Without Triangulation*), a été proposé afin de calculer des décompositions arborescentes adaptées selon des critères donnés. Nous rappelons maintenant ce cadre. Étant donné un graphe $G = (X, C)$ à décomposer, la première étape de *H-TD-WT* (ligne 1 dans l'algorithme 1) calcule un premier cluster, noté E_0, à l'aide d'une heuristique. X' qui représente l'ensemble des sommets déjà considérés est initialisé à E_0 (ligne 2). On notera par $X_1, X_2, \dots X_k$ les composantes connexes du sous-graphe $G[X \backslash E_0]$ induit par la suppression dans G des sommets de E_0 [1]. Chacun de ces ensembles X_i est inséré dans une file d'attente F (ligne 4). Pour chaque élément X_i retiré de F (ligne 6), V_i note l'ensemble des sommets de X' qui sont adjacents à au moins un sommet de X_i (ligne 7). V_i constitue un séparateur dans le graphe G puisque la suppression de V_i dans G déconnecte G (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 (ligne 8) peut être paramétrée à l'aide d'une heuristique. Elle recherche un sous-ensemble de sommets $X_i'' \subseteq X_i$ tel que $X_i'' \cup V_i$ sera un nouveau cluster E_i de la décomposition. Cela peut être garanti s'il existe au moins un sommet v de V_i tel que tous ses voisins dans X_i figurent dans X_i''. Plus précisément, si $N(v, X_i) = \{x \in X_i : \{v, x\} \in C\}$, nous devons garantir l'existence d'un sommet v de V_i avec $N(v, X_i) \subseteq X_i''$. Nous définissons alors un nouveau cluster $E_i = X_i'' \cup V_i$ (ligne 9). Puis, nous rajoutons à X' les sommets de X_i'' (ligne 10), avant de calculer les composantes connexes du sous-graphe issu de la suppression des sommets de E_i dans $G[X_i]$, composantes qui sont alors rajoutées à la file F (ligne 11). Ce processus est répété jusqu'à ce que la file F soit vide.

Algorithme 1 : *H-TD-WT*

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 sous-ensemble $X_i'' \subseteq X_i$ tel qu'il existe au moins un sommet $v \in V_i$ tel que $N(v, X_i) \subseteq X_i''$
9 $E_i \leftarrow X_i'' \cup V_i$
10 $X' \leftarrow X' \cup X_i''$
11 Soient $X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}$ les composantes connexes de $G[X_i \backslash E_i]$
12 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$

Au final, *H-TD-WT* calcule une décomposition arborescente du graphe $G = (X, C)$ sans triangulation en temps polynomial (à savoir en $O(n(n + e))$). Bien sûr, comme pour Min-Fill, aucune garantie n'est offerte quant à l'optimalité de la largeur obtenue. Par contre, dans le cadre de la résolution d'instances (W)CSP, un choix judicieux de paramètres peut conduire à produire des décompositions plus adaptées que celles produites par Min-Fill. Parmi les paramètres envisageables, on peut citer, par exemple, la connexité des clusters, la taille des clusters produits ou encore la taille des séparateurs (c'est-à-dire des intersections entre clusters). Nous considérons, par la suite, les trois déclinaisons suivantes proposées dans le cadre de la résolution du problème de décision CSP et qui visent à rendre la résolution plus efficace :

- H_2 [14] qui garantit que tous les clusters produits sont connexes,
- H_3 [12] qui identifie des parties indépendantes dans le graphe et les sépare aussitôt que possible en effectuant un parcours en largeur à partir des sommets de V_i,
- H_5 [13] qui vise à produire des décompositions dont les séparateurs entre clusters sont de taille au plus S.

1. Le sous-graphe $G[Y]$ de $G = (X, C)$ induit par $Y \subseteq X$ est le graphe (Y, C_Y) où $C_Y = \{\{x, y\} \in C | x, y \in Y\}$.

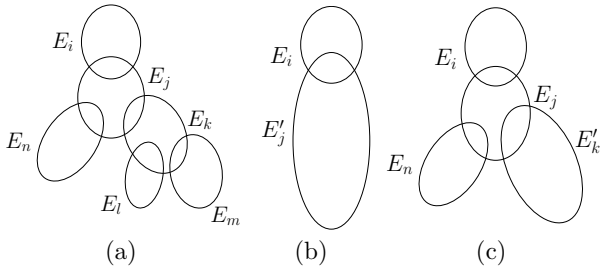


FIGURE 2 – Ensemble de clusters de la décomposition : initial (a) lorsque E_j est fusionné avec sa descendance (cluster E'_j) (b) lorsque E_j est exploité (c).

Une fois une décomposition arborescente calculée, elle peut être exploitée de façon statique comme dans [23, 11, 1] ou de manière dynamique comme présenté dans la section suivante.

3 Exploitation dynamique de la décomposition

Nous visons à exploiter la décomposition arborescente plus intelligemment en évitant de l'utiliser systématiquement. En effet, selon la nature de l'instance à résoudre, sa résolution grâce à des méthodes n'ayant pas recours à une décomposition peut être très efficace. Cela s'explique par les améliorations qu'ont connues ces méthodes notamment en s'associant aux techniques dites *adaptatives*. Ces techniques permettent de faire des choix plus judicieux pendant la résolution en tenant compte de l'état courant de la résolution mais aussi des états précédents. Ce faisant, elles sont capables de guider la recherche vers les parties les plus problématiques. Grâce à elles, par exemple, une heuristique de choix de variables basée sur les conflits va se concentrer sur les variables jugées les plus pertinentes et les plus difficiles. Nous citons à l'appui l'intérêt de guider la recherche grâce à la pondération des contraintes et l'enregistrement des conflits [3], ou grâce à la notion d'activité des variables [18] ou leur impact [19]. Ces techniques profitent aussi de l'absence de contraintes engendrées par l'usage d'une décomposition. En effet, l'exploitation d'une décomposition induit un ordre partiel sur les variables qui impose des restrictions à l'heuristique de choix de variables et limite potentiellement l'intérêt des techniques adaptatives. D'ailleurs, ces techniques peuvent parfois s'avérer inutiles si l'ordre induit par la décomposition est total aboutissant à un choix de variables statique avec tous les inconvénients que cela pourrait engendrer par rapport à un choix de variables dynamique. En plus de leur association avec les techniques adaptatives, ces algorithmes ont connu une amélioration remarquable au niveau de leur stratégie de recherche. Plus précisément, dans [1], la combinaison de deux stratégies de recherche, à savoir le parcours en profondeur (DFS) et le parcours le meilleur d'abord (BFS) a amélioré significativement la résolution des problèmes d'optimisation sous contraintes. Naturellement, si les limitations imposées par la décomposition au niveau de l'heuristique de choix de variables sont très restrictives, l'efficacité pratique de cette hybridation peut s'en retrouver détériorée. En revanche, l'utilisation d'une décomposition peut être parfois très bénéfique pour la résolution. C'est essentiellement le cas des instances qui présentent des propriétés structurelles intéressantes. En particulier, l'enregistrement des informations au niveau des séparateurs entre les clusters permet d'élaguer des parties de l'espace de recherche et de rendre la résolution plus efficace. Dans le cadre de l'optimisation, ces enregistrements permettent de mémoriser, pour chaque sous-problème considéré, les meilleures bornes inférieure et supérieure calculées, ou mieux encore, son optimum. Afin de concilier ces deux points de vue, nous proposons d'exploiter la décomposition dynamiquement d'une façon plus flexible et plus appropriée au vu du contexte courant de la résolution. Cela permet de s'adapter progressivement à la nature de l'instance à résoudre. La forme de dynamicité que nous proposons dans ce cadre est une utilisation de la décomposition vis-à-vis d'un sous-problème justifiée par une stagnation de la recherche sans décomposition. En d'autres termes, la décomposition n'est utilisée que lorsque son utilisation semble pertinente.

L'algorithme BTD+DYN (voir l'algorithme 2) se base sur l'algorithme BTD-DFS proposée dans [11] et repris dans [1]. Les modifications apportées représentent une adaptation de l'algorithme afin de prendre en compte l'exploitation dynamique de la décomposition. Les deux algorithmes se basent sur une décomposition arborescente calculée en amont de la résolution et qui est enracinée en un cluster racine E_r. En ce qui concerne BTD-DFS, la décomposition induit un ordre partiel sur les variables. Si E_j est le cluster courant, le choix de la prochaine variable s'effectue soit parmi les variables non instanciées du cluster E_j, soit, une fois le cluster E_j entièrement instancié, parmi les variables non instanciées d'un cluster fils de E_j (celui-ci étant choisi de manière heuristique parmi tous les fils de E_j). Comme la décomposition ne change pas pendant la résolution, il est en de même pour l'ordre partiel sur les variables. Cependant, BTD+DYN exploite la décomposition différemment. Plus précisément, la décomposition calculée initialement ne sera exploitée pour un sous-problème donné que si la résolution sans décomposition est jugée inefficace. Prenons l'exemple de la décomposition de la figure 2(a) représentée par l'en-

Algorithme 2 : BTD+DYN ($\mathcal{A}, E_i, V, V_{desc}, clb, cub$)

Entrées : L'affectation courante $\mathcal{A}$, le cluster courant E_i, la borne inférieure clb
Entrées-Sorties : L'ensemble V des variables non instanciées de E_i, l'ensemble V_{desc} des variables non instanciées de $Desc(E_i)$, la borne supérieure courante cub

1 **si** $Fusion(E_i)$ **alors**
2 $V' := V_{desc}$
3 **sinon**
4 $V' := V$
5 **si** $V' \neq \emptyset$ **alors**
6 x :=dépiler(V') /* Choisir une variable de V' */
7 Mettre à jour V et V_{desc}
8 a :=dépiler(D_x) /* Choisir une valeur */
9 Appliquer la cohérence locale au sous-problème $P_i|\mathcal{A} \cup \{(x = a)\}$
10 $clb' := max(clb, lb(P_i|\mathcal{A} \cup \{(x = a)\}))$
11 **si** $clb' < cub$ **alors**
12 cub :=BTD+DYN($\mathcal{A} \cup \{(x = a)\}, E_i, V, V_{desc}, clb', cub$)
13 **si** $max(clb, lb(P_i|\mathcal{A})) < cub$ **alors**
14 Appliquer la cohérence locale au sous-problème $P_i|\mathcal{A} \cup \{(x \neq a)\}$
15 $clb' := \max(clb, lb(P_i|\mathcal{A} \cup \{(x \neq a)\}))$
16 **si** $clb' < cub$ **alors**
17 $cub :=$ BTD+DYN $(\mathcal{A} \cup \{(x \neq a)\}, E_i, V, V_{desc}, clb', cub)$
18 **sinon**
19 **si** $\neg Fusion(E_i)$ **alors**
20 $S := Fils(E_i)$
21 /*Résoudre les fils dont l'optimum n'est pas connu */ **tant que** $S \neq \emptyset$ **et** $lb(P_i|A) < cub$ **faire**
22 E_j :=dépiler(S) /* Choisir un cluster fils */
23 **si** $LB_{P_j|\mathcal{A}} < UB_{P_j|\mathcal{A}}$ **alors**
24 $cub' := \min\,(UB_{P_j|\mathcal{A}}, cub - [lb(P_i|\mathcal{A}) - lb(P_j|\mathcal{A})])$
25 $cub'' :=$ BTD+DYN $(\mathcal{A}, E_j, E_j \backslash (E_i \cap E_j), Desc(E_j) \backslash (E_i \cap E_j), lb(P_j|\mathcal{A}), cub')$
26 Mettre à jour $LB_{P_f|\mathcal{A}}$ et $UB_{P_f|\mathcal{A}}$ en se basant sur cub''
27 $cub := \min(cub, w^i_\emptyset + \sum_{E_j \in Fils(E_i)} UB_{P_j|\mathcal{A}})$
28 **sinon**
29 $cub := \min(cub, \sum_{E_j \in Desc(E_i)} w^j_\emptyset)$
30 **retourner** cub

semble de clusters $E = \{E_i, E_j, E_k, E_l, E_m, E_n\}$. Soit E_j le cluster courant et $\mathcal{A}$ l'affectation courante sur $E_j \cap E_i$. On s'intéresse au sous-problème $P_j|\mathcal{A}$ enraciné en E_j induit par l'affectation $\mathcal{A}$ du séparateur $E_j \cap E_i$ avec E_i le cluster parent de E_j. La résolution du sous-problème $P_j|\mathcal{A}$ n'utilisera pas forcément la décomposition, soit l'ensemble des clusters $\{E_j, E_k, E_l, E_m, E_n\}$. En effet, au début, la résolution est basée sur le cluster E'_j résultant de la fusion du cluster E_j avec ses descendants E_k, E_l, E_m et E_n. Cette fusion consiste en une mise en commun de toutes les variables de la descendance de E_j (voir la figure 2(b)). Ainsi E'_j contient toutes les variables de la descendance de E_j. En ce qui concerne l'heuristique de choix de variables, elle acquiert une liberté totale quant au choix des variables suivantes sur la descendance de E_j. À ce niveau, deux cas se présentent :

— Le sous-problème $P_j|\mathcal{A}$ est facilement résolu en se basant sur E'_j. Dans ce cas, l'exploitation de la décomposition ne semble pas utile.
— La résolution de $P_j|\mathcal{A}$ est au contraire inefficace. Dans ce cas, l'exploitation de la décomposition semble judicieuse. Le cluster E_j est alors réexploité et le même raisonnement est répété au niveau du cluster E_k qui sera au début considéré fusionné avec sa descendance formant le cluster E'_k comme le montre la figure 2(c).

Quant aux clusters feuilles (c'est-à-dire n'ayant pas de fils, comme E_n), BTD+DYN se comporte comme BTD-DFS. À noter que ce raisonnement est refait pour chaque nouvelle affectation de $E_i \cap E_j$. Ce choix est motivé par le fait qu'une affectation $\mathcal{A}$ du séparateur $E_i \cap E_j$ induit un sous-problème différent de celui induit par une affectation $\mathcal{A}'$. À noter aussi, qu'au niveau du cluster racine E_r, BTD+DYN se comporte comme l'algorithme de base ayant toute la liberté concernant le choix des variables du problème. Finalement, la décomposition initiale est utilisée complètement (tous ses clusters sont exploités) si à tous les niveaux BTD+DYN décide d'exploiter le cluster lui-même plutôt que sa descendance.

L'algorithme BTD+DYN prend en paramètres l'affectation courante $\mathcal{A}$, le cluster courant E_i, l'ensemble V des variables non affectées de E_i, l'ensemble V_{desc} des variables non affectées de la descendance $Desc(E_i)$ de E_i (c'est-à-dire l'ensemble des clusters situés dans le sous-arbre enraciné en E_i, E_i inclus), et les bornes inférieure et supérieure courantes clb et cub. Il vise à calculer l'optimum du problème enraciné en E_i et induit par l'affectation $\mathcal{A}$. Deux cas sont possibles :

— Si la valeur de cub retournée est strictement inférieure à celle de cub en entrée, alors cub est l'optimum correspondant à ce problème.
— Sinon, cub définit une borne inférieure de l'optimum.

L'appel initial est BTD+DYN($\emptyset$, E_r, V_r, $V_{r_{desc}}$, $lb(P_r|\emptyset)$, k) avec $V_{r_{desc}}$ l'ensemble des variables de la descendance de E_r, $lb(P_r|\emptyset)$ la borne inférieure initiale déduite grâce à l'application de la cohérence locale au problème initial et k le coût maximal. Notons que tout au long de l'algorithme, $w^i_\emptyset$ désigne la borne inférieure localisée relative au cluster E_i. Comme BTD-DFS, BTD+DYN suppose que le problème donné en entrée est cohérent selon la cohérence locale utilisée et renvoie son optimum. Les lignes 5 à 17 de BTD+DYN visent à instancier les variables de V' comme le ferait BTD-DFS pour les variables de V. Un couple (variable, valeur), (x, a), est sélectionné selon les heuristiques de choix de variables et de valeurs employées. Comme le

type de branchement utilisé est binaire, ce choix se décline en deux branches $x = a$ (ligne 9) et $x \neq a$ (ligne 14). Pour chaque branche, la cohérence locale est appliquée au sous-problème enraciné en E_i et induit par l'affectation courante permettant d'obtenir une borne inférieure *lb*. Si le maximum *clb′* entre la borne inférieure *lb*, déduite par la cohérence locale, et la borne inférieure courante *clb*, est toujours inférieure à *cub*, la recherche continue ; sinon le sous-arbre correspondant est élagué. Les lignes 20 à 27 de BTD+DYN résolvent les sous-problèmes enracinés en chaque cluster fils E_j de E_i de la même façon que BTD-DFS. Le sous-problème $P_j|\mathcal{A}$ n'est exploré que si son optimum n'est pas déjà connu. D'ailleurs, BTD-DFS et BTD+DYN mémorise pour chaque sous-problème $P_j|\mathcal{A}$ deux valeurs, notées $LB_{P_j|\mathcal{A}}$ et $UB_{P_j|\mathcal{A}}$, repésentant respectivement les meilleures bornes inférieure et supérieure connues pour $P_j|\mathcal{A}$. Si $LB_{P_j|\mathcal{A}} = UB_{P_j|\mathcal{A}}$, l'optimum de P_j est déjà calculé. L'appel récursif correspondant au fils E_j (ligne 25) exploite une borne supérieure initiale non triviale calculée à la ligne 24 comme expliqué dans [11]. Il est suivi par une mise à jour de $LB_{P_j|\mathcal{A}}$ et de $UB_{P_j|\mathcal{A}}$ (ligne 26). L'exploration de tous les clusters fils permet enfin de mettre à jour *cub*. Les similitudes entre BTD-DFS et BTD+DYN rappelées, nous nous intéressons maintenant aux modifications faites. La fonction $Fusion$ représente l'heuristique chargée de faire le choix d'exploiter le cluster E_i ou sa descendance E'_i. L'appel à $Fusion$ avec le cluster E_i en entrée renvoie vrai si et seulement si E'_i est exploité. Sinon, le cluster E_i est exploité et la liberté de choix de variables est restreinte aux variables de V. L'un des deux paramètres V ou V_{desc} est effectivement utilisé selon que l'on exploite E_i ou E'_i. Le choix entre V et V_{desc} est fait dans les lignes 1 à 4 et est retenu dans V'. Ils sont mis à jour convenablement dans la ligne 7. Si $V' = \emptyset$ et que E_i n'est pas fusionné avec sa descendance, BTD+DYN se comporte comme BTD-DFS (lignes 20-27). Si $V' = \emptyset$ et que E_i est fusionné avec sa descendance, BTD+DYN n'a plus de variables à instancier vu que toutes les variables de la descendance de E_i sont déjà affectées. Dans ce cas, BTD+DYN met à jour *cub* (ligne 29) en se référant à la borne inférieure localisée $w^j_\emptyset$ de chaque cluster E_j appartenant à $Desc(E_i)$. Si $V' \neq \emptyset$ (lignes 5-17), BTD+DYN se comporte de la même façon indépendamment du choix de l'exploitation de E_i ou de sa descendance E'_i en essayant d'instancier les variables de V'.

L'algorithme BTD+DYN est paramétrable par l'heuristique $Fusion$ qui se charge de décider de passer de l'exploitation du cluster E'_i à E_i, si E_i est le cluster courant (nous en proposons une dans la partie expérimentale). Un choix pertinent de cette heuristique est, bien sûr, essentiel pour l'amélioration de la résolution.

L'exploitation dynamique de la décomposition induit un changement au niveau des complexités en temps et en espace. Le comportement de BTD+DYN pouvant aller d'un BTD-DFS standard à un classique DFS, il en résulte que la complexité temporelle de BTD+DYN est comprise entre $O(exp(w^+ + 1))$ et $O(exp(n))$ avec w^+ la largeur de la décomposition calculée en amont de la résolution. Toutefois, il est possible de limiter cette complexité en utilisant une heuristique convenable qui se charge d'interdire l'exploitation des clusters de la descendance à des profondeurs faibles, comme dans le cas du cluster racine par exemple. Au niveau de la complexité en espace, elle est exponentielle en fonction de la taille du plus grand séparateur exploité au niveau de la décomposition. Donc, dans le pire des cas, la complexité en espace est la même que celle de BTD-DFS si le plus grand séparateur de la décomposition est utilisé.

4 Expérimentations

Dans cette section, nous évaluons d'une part l'intérêt pratique du cadre de calcul de décompositions *H-TD-WT* et d'autre part l'apport de l'exploitation dynamique des décompositions dans le cadre de la résolution des problèmes d'optimisation sous contraintes. Mais, au préalable, nous décrivons le protocole expérimental suivi.

4.1 Protocole expérimental

Nous considérons les algorithmes HBFS et BTD-HBFS introduits dans [1] et exploitons leurs implémentations fournies dans *Toulbar2*[2]. HBFS consiste en une hybridation de la stratégie de recherche qui combine à la fois les avantages du parcours en profondeur (DFS) et les avantages du parcours le meilleur d'abord (BFS). HBFS est alors capable de donner à tout moment une borne inférieure de l'optimum comme BFS mais aussi une borne supérieure comme DFS. Son intégration avec BTD permet d'exploiter ses caractéristiques au sein de chaque cluster améliorant ainsi le comportement global de ce dernier.

En ce qui concerne les décompositions, nous considérons Min-Fill en tant qu'heuristique de l'état de l'art en utilisant son implémentation fournie dans *Toulbar2*. Une deuxième version de Min-Fill est aussi utilisée et serait référencée par Min-Fill^{r_4}. Elle se distingue de Min-Fill par l'absence de séparateurs de taille supérieure à 4. En effet, chaque cluster de la décomposition calculée par Min-Fill partageant plus de quatre variables avec son père est fusionné avec celui-ci (son

2. Présent dans le répertoire git à `https://mulcyber.toulouse.inra.fr/projects/toulbar2/`

utilisation avec BTD-HBFS correspond à l'algorithme BTD-HBFS^{r_4} dans [1]). Du côté de H-TD-WT, nous retenons les décompositions H_2 qui garantit d'obtenir des clusters connexes, H_3 qui permet de calculer des clusters ayant plusieurs clusters fils et H_5 pour sa maîtrise de la taille des séparateurs. L'heuristique H_5 est déclinée en deux variantes notées H_5^{25} et $H_5^{5\%}$. H_5^{25} limite la taille maximale des séparateurs à 25. Quant à $H_5^{5\%}$, elle exploite une taille maximale de séparateurs dépendant de l'instance, fixée à 5% du nombre de variables de l'instance dans la limite d'au moins 4 variables et au plus 50. Les décompositions H_i sont calculées au sein de notre propre bibliothèque et communiquées à *Toulbar2* par l'intermédiaire d'un fichier. Notons qu'à l'heure actuelle, compte tenu de leur efficacité pratique [1], HBFS et BTD-HBFS avec Min-Fill^{r_4} peuvent être considérés comme étant les références en tant qu'algorithmes de résolution des instances WCSP respectivement sans et avec exploitation de la structure.

L'exploitation dynamique de la décomposition repose sur une heuristique ($\mathcal{F}$) qui se charge de décider d'exploiter E_i au lieu d'exploiter toute la descendance E_i' de E_i. L'heuristique $\mathcal{F}$ se base sur le retour d'information fait par BTD-HBFS. En effet, chaque appel de BTD-HBFS sur un sous-problème $P_i|\mathcal{A}$ prend en entrée une borne inférieure *clb* et une borne supérieure *cub*. Si, dans la limite du nombre de backtracks autorisé, BTD-HBFS ne réussit à améliorer aucune de ces deux bornes, $\mathcal{F}$ considère que la résolution de $P_i|\mathcal{A}$ n'avance pas et incrémente un compteur qui lui est propre. Lorsque le compteur relatif à $P_i|\mathcal{A}$ atteint une certaine limite (5 dans nos expérimentations), on exploite par la suite E_i en stoppant l'exploitation de E_i'.

En ce qui concerne la configuration de *Toulbar2*, un pré-traitement reposant sur la cohérence d'arc est appliqué via VAC (pour *Virtual Arc Consistency* [7]) en plus de l'application de l'algorithme MSD (pour *Min Sum Diffusion*) avec 1 000 itérations. La cohérence locale est ensuite maintenue pendant la résolution grâce à EDAC (Existential Directional Arc Consistency [10]). L'heuristique de choix de variables est dom/wdeg [3] associée à l'heuristique du dernier conflit [17]. Pour les algorithmes comme BTD, le cluster racine choisi est le plus grand cluster (pour Min-Fill) ou le cluster maximisant le ratio entre le nombre de contraintes du cluster et sa taille (pour les autres décompositions). 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.

Nous avons utilisé le benchmark d'instances disponible à l'adresse `http://genoweb.toulouse.inra.fr/~degivry/evalgm`. Il est composé de plus de 3 000 instances incluant entre autres des modèles graphiques stochastiques de l'évaluation UAI de 2008 et 2010, des instances de la compétition MiniZinc 2012 et 2013 et des instances de la compétition PIC 2011. Nous avons écarté les instances résolues pendant la phase de prétraitement pour obtenir au final un benchmark composé de 2 444 instances. Pour chaque instance, chacun des algorithmes de résolution considérés dispose de 20 minutes (ce temps incluant, le cas échéant, le temps requis pour calculer une décomposition) et 16 Go de mémoire.

4.2 H-TD-WT comparée à Min-Fill

Nous comparons, dans cette partie, le comportement de BTD-HBFS en fonction des différentes décompositions exploitées. Au niveau du calcul de décompositions, nous constatons que le calcul des décompositions avec H_i ($i \in \{2, 3, 5\}$) est nettement plus rapide qu'avec Min-Fill. Plus précisément, le temps total de calcul de décompositions ne dépasse pas 1 200 s pour les 2 431 instances décomposées tandis que Min-Fill requiert 19 044 s pour décomposer 2 415 instances. Les tables 1 et 2 fournissent le nombre d'instances résolues et le temps d'exécution cumulé pour BTD-HBFS avec les décompositions de l'état de l'art et les décompositions de H-TD-WT. Tout d'abord, notons que Min-Fill résout le plus petit nombre d'instances en 20 minutes (seulement 1 712 instances). Cela montre que, malgré la difficulté du problème de l'optimisation sous contraintes, les heuristiques de décompositions cherchant uniquement à minimiser la taille des clusters ne sont pas les plus efficaces. En effet, vu le faible nombre de variables propres (c'est-à-dire de variables appartenant à un cluster et pas à son cluster père) dans chaque cluster (souvent une seule variable) l'heuristique de choix de variables est quasiment inutile et l'ordre d'instanciation de variables est presque statique. Les résultats montrent aussi que les autres paramètres ont plus d'impacts comme la connexité des clusters (H_2), le nombre de fils d'un cluster (H_3) ou la taille des séparateurs (Min-Fill^{r_4} et H_5). Le fait de borner la taille des séparateurs semble jouer un rôle crucial dans l'augmentation de l'efficacité de la résolution. En effet, si BTD-HBFS avec H_2 et H_3 résout respectivement 1 945 et 1 989 instances, les décompositions dont la taille de séparateurs est bornée permettent de résoudre plus de 2 000 instances. La comparaison de Min-Fill^{r_4} avec H_5 montre cependant que H_5 permet à BTD-HBFS de résoudre plus d'instances notamment avec $H_5^{5\%}$ qui résout 2 029 instances contre 2 000 instances pour Min-Fill^{r_4}. En outre, BTD-HBFS associé à Min-Fill^{r_4} requiert 98 861 s en temps d'exécution cumulé contre seulement 58 792 s pour $H_5^{5\%}$. Notons enfin que ces résultats sont cohérents avec ceux obte-

Algorithme	*Min-Fill*		*Min-Fill*^{r_4}	
	#rés.	temps	#rés.	temps
`BTD-HBFS`	1 712	26 291	2 000	98 861
`BTD-HBFS+DYN`	1 905	45 450	2 019	74 159

TABLE 1 – Nombre d'instances résolues et temps d'exécution en secondes pour BTD-HBFS et BTD-HBFS+DYN selon les décompositions de l'état de l'art.

Algorithme	H_2		H_3		H_5^{25}		$H_5^{5\%}$	
	#rés.	temps	#rés.	temps	#rés.	temps	#rés.	temps
`BTD-HBFS`	1 945	74 686	1 989	57 254	2 006	56 553	2 029	58 792
`BTD-HBFS+DYN`	2 023	50 445	2 023	56 978	2 039	52 304	2 038	60 674

TABLE 2 – Nombre d'instances résolues et temps d'exécution en secondes pour BTD-HBFS et BTD-HBFS+DYN selon les décompositions de H-TD-WT.

nus pour le problème de décision CSP dans [12].

4.3 BTD-HBFS contre HBFS

Nous comparons BTD-HBFS à HBFS. Nous retenons, dans cette partie, la décomposition $H_5^{5\%}$ qui permet d'obtenir les meilleurs résultats avec BTD-HBFS par rapport aux autres décompositions. BTD-HBFS résout plus d'instances que HBFS, à savoir 2 029 instances contre 2 017 instances pour HBFS. En plus, BTD-HBFS ne nécessite que 58 792 s en temps cumulé d'exécution contre 84 555 s pour HBFS. Cela peut s'expliquer essentiellement par les enregistrements effectués par BTD-HBFS au niveau des séparateurs, à savoir une borne inférieure et une borne supérieure de l'optimum d'un sous-problème. De plus, BTD-HBFS détecte les indépendances entre les sous-problèmes, ce qui n'est pas le cas de HBFS.

4.4 Évaluation de la dynamicité

Nous évaluons à présent BTD-HBFS+DYN. Les deux tables 1 et 2 montrent, que quelle que soit la décomposition exploitée, BTD-HBFS+DYN résout plus d'instances que BTD-HBFS. L'augmentation du nombre d'instances résolues peut être considérable. C'est notamment le cas pour Min-Fill pour lequel une exploitation dynamique permet de résoudre 1 905 instances contre 1 712 instances dans le cadre d'une exploitation statique. L'utilisation de H_2 et H_3 avec BTD-HBFS+DYN permet de résoudre 2 023 instances contre 1 945 et 1 989 résolues par BTD-HBFS. L'exploitation de Min-Fill^{r_4} et H_5 est aussi améliorée avec BTD-HBFS+DYN par rapport à BTD-HBFS. En particulier, H_5^{25} permet de résoudre 2 039 instances, ce qui constitue le plus grand nombre d'instances résolues parmi toutes les combinaisons d'algorithme de résolution et de décomposition présentées. L'augmentation du nombre d'instances résolues est souvent accompagnée d'une diminution du temps total d'exécution, sauf pour Min-Fill, mais cela s'explique naturellement par le nombre d'instances supplémentaires résolues par BTD-HBFS+DYN par rapport à BTD-HBFS. En outre, nous nous intéressons à la comparaison de H_5^{25} à l'heuristique de l'état de l'art Min-Fill^{r_4} avec BTD-HBFS+DYN et aussi à la comparaison de BTD-HBFS+DYN basé sur H_5^{25} à HBFS. La figure 3(a) présente une comparaison des temps d'exécution de BTD-HBFS+DYN avec H_5^{25} à BTD-HBFS+DYN avec Min-Fill^{r_4} tandis que la figure 3(b) le compare à HBFS. Dans les deux cas, BTD-HBFS+DYN associé à H_5^{25} prouve clairement son intérêt pratique. Pour une comparaison plus équitable des temps cumulés d'exécution, nous nous basons sur le benchmark des instances résolues à la fois par BTD-HBFS+DYN avec Min-Fill^{r4} et par BTD-HBFS+DYN avec H_5^{25}. Il compte 2 013 instances résolues par BTD-HBFS+DYN avec Min-Fill^{r4} en 71 249 s contre seulement 41 372 s par BTD-HBFS+DYN avec H_5^{25}. Nous procédons, de même, pour comparer HBFS et BTD-HBFS+DYN avec H_5^{25}. Nous obtenons alors un total de 2 008 instances résolues en 79 020 s par HBFS contre seulement 43 914 s par BTD-HBFS+DYN. Le couplage de BTD-HBFS+DYN avec H_5^{25} assure notamment que les méthodes basées sur la décomposition peuvent être compétitives vis-à-vis des méthodes n'exploitant pas la décomposition comme HBFS. Tous ces résultats montrent que la dynamicité exploitée au niveau de la décomposition semble permettre à BTD-HBFS+DYN de mieux s'adapter à la nature de l'instance et d'éviter de se baser sur une décomposition lorsqu'une résolution sans décomposition s'avère plus efficace que ce soit globalement ou localement.

Nous focalisons maintenant nos observations sur HBFS et BTD-HBFS+DYN associé à H_5^{25}. Nous écartons à ce stade les instances faciles, voire parfois triviales (c'est-à-dire celles résolues en moins de 10 s

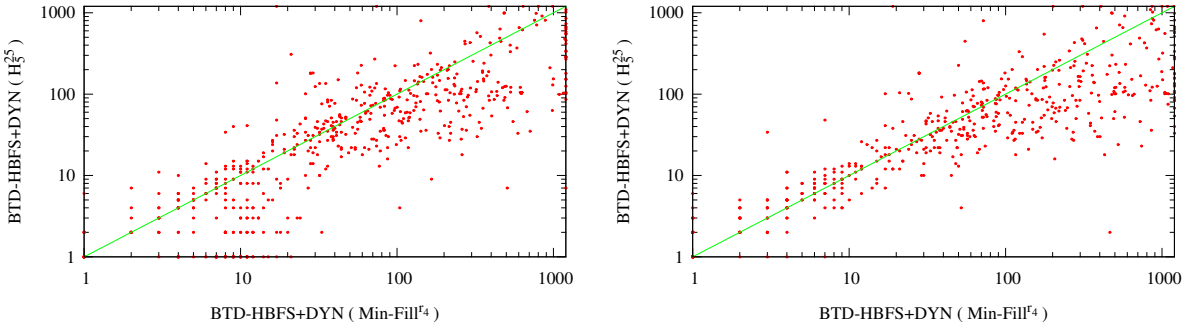


FIGURE 3 – Comparaison des temps d'exécution en secondes de BTD-HBFS+DYN avec H_5^{25} à BTD-HBFS+DYN avec Min-Fill^{r_4} (a) ou à HBFS (b) pour les 2 444 instances.

par HBFS). Notons que ces instances sont aussi facilement résolues par BTD-DYN+HBFS vu qu'au niveau du cluster racine BTD-DYN+HBFS se comporte comme HBFS (lorsqu'on exploite la descendance relative au cluster racine), avec aussi l'avantage de pouvoir distinguer les composantes connexes d'une instance ce qui n'est pas le cas de HBFS. Parmi les 794 instances restantes, pour 279 instances au moins un cluster feuille de la décomposition est exploité. Autrement dit, il existe au moins une branche de la décomposition qui est entièrement exploitée (au sens de l'ensemble de clusters d'origine de la décomposition). Pour 423 instances, la décomposition n'est jamais exploitée. Cela signifie que BTD-DYN+HBFS se comporte comme HBFS. Pour les instances restantes, la décomposition est exploitée jusqu'à une certaine profondeur sans pour autant atteindre un cluster feuille de la décomposition. Donc, en pratique, BTD-HBFS+DYN peut se comporter simplement comme HBFS ou faire des choix d'exploitation des clusters d'origine de la décomposition (au lieu de leur descendance) jusqu'à atteindre le comportement de BTD-HBFS.

Enfin nous comparons les bornes supérieures rapportées par HBFS et BTD-HBFS+DYN dans le cas de dépassement du temps limite. Parmi les 304 instances qui ne sont pas résolues par HBFS, ni par BTD-HBFS+DYN, pour 252 instances, la borne supérieure calculée par BTD-HBFS+DYN est strictement inférieure à celle de HBFS contre seulement 52 instances pour HBFS. Cela montre que même lorsque l'instance n'est pas résolue, BTD-HBFS+DYN est capable de donner des approximations de meilleure qualité que HBFS.

5 Conclusion

Les décompositions arborescentes ont déjà été exploitées avec succès pour résoudre des instances du problème WCSP. Pour autant, leur potentiel n'a pas été pleinement exploité notamment du fait de la qualité des décompositions calculées. En effet, à l'image de ce que fait Min-Fill, l'heuristique de référence, leur calcul vise souvent à minimiser la taille des clusters (afin de minimiser la borne de complexité théorique en temps) au détriment de l'efficacité pratique de la résolution. Aussi, pour y remédier, d'une part, nous avons exploité le cadre de décomposition H-TD-WT que nous avons auparavant introduit pour résoudre des instances du problème de décision CSP. Ce cadre vise à calculer des décompositions beaucoup plus rapidement mais aussi à capturer des propriétés intéressantes du point de vue de la résolution comme la maîtrise de la taille des séparateurs avec l'heuristique H_5. L'utilisation conjointe de H_5 avec BTD-HBFS a prouvé l'intérêt de H-TD-WT. En effet, elle a non seulement permis d'améliorer les performances obtenues par BTD-HBFS associé à Min-Fill mais a aussi conduit à surclasser HBFS. D'autre part, nous avons proposé une extension de BTD-HBFS, à savoir BTD-HBFS+DYN, qui exploite la décomposition d'une façon dynamique. L'idée consiste à n'utiliser la décomposition sur un sous-problème que lorsque la résolution sans décomposition semble difficile. Cela évite de restreindre inutilement la liberté de l'heuristique de choix de variables pour des sous-problèmes « faciles ». En pratique, nous avons montré que BTD-HBFS+DYN améliore BTD-HBFS, quelle que soit la décomposition utilisée, en nombre total d'instances et en temps. Aussi, grâce à cette extension, les algorithmes basés sur une décomposition présentent davantage d'intérêt par rapport à ceux non basés sur une décomposition.

Plusieurs extensions de ce travail semblent pertinentes. Par exemple, le calcul de la décomposition fait en amont de la résolution peut, à son tour, être fait dynamiquement. Cela semble d'autant plus intéressant que la décomposition n'est pas forcément exploitée avec BTD-HBFS+DYN. De plus, cela permettrait aussi de calculer des décompositions mieux adaptées en vue de la résolution que celles calculées uniquement sur la base de critères structurels. Au-delà, d'autres heuristiques d'exploitation dynamique de la décomposition sont envisageables.

Remerciements

Nous tenons à remercier les membres du projet *Toulbar2*, et plus particulièrement Simon de Givry, pour leur disponibilité et leur aide.

Références

[1] D. Allouche, S. de Givry, G. Katsirelos, T. Schiex, and M. Zytnicki. Anytime hybrid best-first search with tree decomposition for weighted csp. In *CP*, pages 12–29, 2015.

[2] S. Arnborg, D. Corneil, and A. Proskuroswki. Complexity of finding embeddings in a k-tree. *SIAM Journal of Disc. Math.*, 8 :277–284, 1987.

[3] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *ECAI*, pages 146–150, 2004.

[4] C. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio Link Frequency Assignment. *Constraints*, 4 :79–89, 1999.

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

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

[7] M. C. Cooper, S. de Givry, M. Sánchez, T. Schiex, and M. Zytnicki. Virtual arc consistency for weighted csp. In *AAAI*, pages 253–258, 2008.

[8] M. C. Cooper, S. de Givry, and T. Schiex. Optimal Soft Arc Consistency. In *IJCAI*, pages 68–73, 2007.

[9] S. de Givry, F. Heras, J. Larrosa, and M. Zytnicki. Existential arc consistency : closer to full arc consistency in weighted CSPs. In *IJCAI*, 2005.

[10] S. de Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency : Getting closer to full arc consistency in weighted csps. In *IJCAI*, pages 84–89, 2005.

[11] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *AAAI*, pages 22–27, 2006.

[12] P. Jégou, H. Kanso, and C. Terrioux. An Algorithmic Framework for Decomposing Constraint Networks. In *ICTAI*, pages 1–8, 2015.

[13] P. Jégou, H. Kanso, and C. Terrioux. Improving Exact Solution Counting for Decomposition Methods. In *ICTAI*, pages 327–334, 2016.

[14] P. Jégou and C. Terrioux. Combining Restarts, Nogoods and Decompositions for Solving CSPs. In *ECAI*, pages 465–470, 2014.

[15] P. Jégou and C. Terrioux. Tree-decompositions with connected clusters for solving constraint networks. In *CP*, pages 407–423, 2014.

[16] A. Koster. *Frequency Assignment - Models and Algorithms*. PhD thesis, University of Maastricht, Novembre 1999.

[17] C. Lecoutre, L. Saïs, S. Tabary, and V. Vidal. Reasoning from last conflict (s) in constraint programming. *Artificial Intelligence*, 173(18) :1592–1614, 2009.

[18] L. Michel and P. Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *CP-AI-OR*, pages 228–243, 2012.

[19] P. Refalo. Impact-based search strategies for constraint programming. In *CP*, pages 557–571, 2004.

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

[21] D. J. Rose. Triangulated Graphs and the Elimination Process. *Journal of Mathematical Analysis and Application*, 32 :597–609, 1970.

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

[23] C. Terrioux and P. Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *CP*, pages 709–723, 2003.

Comparaison de structures protéiques : résolution de la maximisation du recouvrement de cartes de contacts par solveur LMC

Olivier Gérard * **Chu-Min Li** **Corinne Lucet**

[1] Université Picardie Jules Verne, Laboratoire Modélisation, Information et Sytèmes
33 rue St Leu, 80000 Amiens
olivier.gerard@etud.u-picardie.fr {chu-min.li, corinne.lucet}@u-picardie.fr

Résumé

L'étude de la similarité entre deux structures protéiques a fait l'objet de nombreuses recherches ces dernières décennies. Il est admis que des protéines présentant des similarités de structure fortes ont très probablement des fonctions similaires. Aujourd'hui, l'enjeu de ces recherches est de trouver un moyen d'automatiser la comparaison et la classification des protéines. Pour atteindre cet objectif, différentes méthodes ont été mises au point et présentées dans la littérature. Parmi celles-ci, nous nous intéressons plus particulièrement à la méthode dite de "maximisation du recouvrement des cartes de contacts", qui peut être transposée en un problème de recherche de clique maximum dans un graphe. Dans cet article nous détaillerons les étapes de cette transposition, de la modélisation des protéines jusqu'à la résolution du problème par un algorithme de recherche de clique maximum appelé LMC qui permet de traiter des graphes de très grande taille, ce qui est essentielle pour aborder les benchmarks relatifs à ce problème.

Abstract

The study of proteins structures similarities has been the subject of numerous works over the past decades. It is admitted that proteins sharing strong similarities will have similar functions. Nowadays, the goal of this study is to find a way to automate the comparison and the classification of proteins. In order to reach this goal, many methods have been set up over the years. Among them, we are especially interested in the Contact Map Overlap maximization, which transforms the comparison of proteins structures similarities into the search of a maximum clique (LMC) research in a graph. In this paper we will detail the steps of this transformation, from the modelling of proteins to the resolution of the problem by our maximum clique finder algorithm LMC. This algorithm is able to treat very large graphs, essential to deal with the benchmarks related to this problem.

*Stagiaire Master I

1 Introduction

Les protéines sont composées de chaînes d'acides aminés, liés par des liens peptidiques dans l'ordre induit par le gène qui code ces protéines. Il existe vingt acides aminés différents qui entrent dans leur composition. Lorsque les protéines sont transcrites, elles peuvent sous certaines conditions se plier, formant une structure tri-dimensionnelle propre à chaque type de protéine. De nombreux biologistes estiment que leurs fonctions sont liées à cette structure, et qu'il est donc possible de classer les protéines en fonction de celle-ci. Ce point de vue s'avère particulièrement intéressant car il suppose qu'il est possible de prédire les fonctions de certaines protéines en comparant simplement leurs structures avec celles de protéines dont on connaît les fonctions.

Il existe différentes méthodes de comparaison des structures, telles que la méthode RMSD (Root Mean Squared Deviation) proposée par W. Kabsch dans [9], la méthode de recouvrement des cartes de contact, ou CMO (Contact Map Overlap), proposée par A. Godzik dans [6], ou encore une méthode proposée plus récemment par N. Malod-Dognin dans [10] basée sur les distances internes appelée DAST (Distance-based Alignment Search Tool). Dans ce travail, nous nous intéressons particulièrement à la méthode CMO, largement utilisée dans la littérature pour sa flexibilité, son insensibilité aux translations et aux rotations ou encore ses scores de similarités en accord avec les classifications structurales telles que SCOP ou CATH [4].

Le problème induit par la méthode CMO consiste à calculer un taux de similarité entre deux structures de protéines en se basant sur les distances euclidiennes entre différents composants. Il s'agit d'un problème NP-difficile [7]. La littérature fait état de nombreuses études de ce problème, qui ont permis de développer des méthodes complètes ou incomplètes. Parmi celles-ci on trouve par exemples des algorithmes basés sur la programmation linéaire et la relaxation lagrangienne [1]. D'autres approches ont transposé le problème en un autre problème NP-diffile, la recherche d'une clique maximum dans un graphe [5, 11, 13]. La particularité des graphes qui modélisent le problème par la méthode CMO est leur très grande taille et leur densité relativement faible. Pour cette raison, nous proposons d'utiliser l'algorithme exact de recherche de clique maximum nommé LMC [8] qui permet de trouver et prouver une clique maximum dans des graphes de plusieurs dizaines de millions de sommets en un temps raisonnable. Pour trouver et prouver cette clique maximum, LMC développe un arbre de recherche en suivant un schéma branch-and-bound, et utilise un prétraitement efficace pour éliminer les sommets qui ne peuvent pas appartenir à une clique maximum et un raisonnement MaxSAT incrémental pour minimiser le nombre de branches dans l'espace de recherche.

Dans cet article, nous détaillerons la définition du problème, puis sa résolution ; des fichiers représentants les protéines jusqu'à la formation du graphe, permettant ainsi l'utilisation de l'algorithme LMC.

2 Recouvrement des cartes de contacts

Une protéine est une chaîne d'acides aminés, que l'on peut représenter par une séquence $(A_1, A_2, \ldots, A_n)$. Ces acides aminés sont plus ou moins proches les uns des autres dans la structure tertiaire de la protéine, qui correspond au repliement de la chaîne polypeptidique dans l'espace. Dans cette structure, et indépendamment de la séquence codant la protéine, des acides aminés non consécutifs peuvent se retrouver proches en fonction des courbes qu'elle décrit. Cette situation est illustrée par la Figure 1, qui représente une protéine fictive composée de sept acides aminés, dont deux d'entre eux (A_2 et A_6) se retrouvent proches alors qu'ils ne sont pas consécutifs. L'objectif de la méthode par maximisation du recouvrement des cartes de contacts est de mesurer la similarité des structures tertiaires des protéines en étudiant la proximité de ces acides aminés non consécutifs.

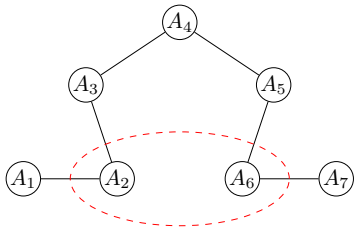


FIGURE 1 – Représentation schématique d'une structure de protéine.

2.1 Définition du problème

La structure tri-dimensionnelle d'une protéine peut être modélisée par une carte de contact qui représente les relations de proximités des acides aminés. Une carte de contacts peut être modélisée par un graphe non orienté G = (V, E) dont les sommets représentent les acides aminés de la protéine et dans lequel il existe une arête $(i, j) \in E$ si et seulement si les deux acides aminés A_i et A_j ne sont pas consécutifs dans la séquence et si la distance euclidienne de leurs carbones alpha est inférieure à un seuil préalablement défini. La Figure 2 représente la carte de contact d'une protéine hypothétique A. Dans cet exemple, le premier acide aminé est en contact avec le troisième et le cinquième, mais pas avec le second ni avec le quatrième. On note que la valeur de ce seuil peut varier de manière importante d'une étude à l'autre dans la littérature, allant de 4Å jusqu'à 20Å [3, 12].

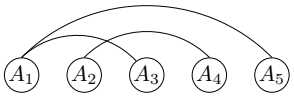


FIGURE 2 – Carte de contact d'une protéine C^a.

À partir des cartes de contacts de deux protéines A et B, leur similarité est mesurée par le nombre maximum de recouvrements entre les cartes de contacts de A et de B . Un recouvrement est une paire de contacts telle que les deux extrémités d'un contact dans une première protéine sont respectivement alignés avec les extrémités d'un contact dans une seconde protéine. Un alignement faisable doit respecter deux restrictions : la restriction d'exclusivité et la restriction d'ordre. La première impose qu'un acide aminé d'une protéine ne peut être lié à plus d'un acide aminé dans l'autre protéine. La seconde impose que les alignements ne brisent pas l'ordre des acides aminés. Prenons quatre acides aminés A_i, A_j, A_k et A_l d'une protéine A et quatre acides aminés $B_{i'}$, $B_{j'}$, $B_{k'}$ et $B_{l'}$ d'une protéine B. Deux contacts (A_i, A_j) et (A_k et A_l) de A peuvent recouvrir deux contacts ($B_{i'}$, $B_{j'}$) et ($B_{k'}$, $B_{l'}$) de B

si et seulement si i, j, k et l sont dans le même ordre que i', j', k' et l', et que l'alignement induit : A_i à $B_{i'}$, A_j à $B_{j'}$, A_k à $B_{k'}$ et A_l à $B_{l'}$ est une bijection. Par exemple, les recouvrements n'est pas possible si $j < k$ et $j' > k'$ ou si $i = k$ et $i' \neq k'$. La solution optimale du problème est celle comportant le plus de recouvrements, tout en respectant les deux restrictions expliquées précédemment. La Figure 3 représente une solution optimale pour deux protéines hypothétiques A (identique à celle utilisée précédemment) et B. Elle est optimale car on ne peut pas trouver plus de recouvrements tout en respectant les restrictions d'ordre et d'exclusivité. Les contacts (A_1, A_3) et (A_1, A_5) recouvrent les contacts (B_1, B_2) et (B_1, B_4) respectivement. Les recouvrements induit l'alignement des acides aminés. Les arêtes plus épaisses représentent les recouvrements des contacts des protéines et les traitillés représentent l'alignement des acides aminés. Les deux restrictions impliquent que les les traitillés ne se croisent pas.

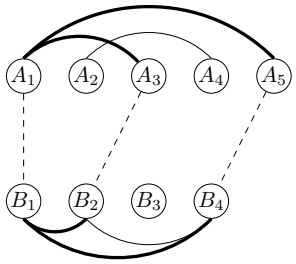


FIGURE 3 – Solution optimale au problème de maximisation du recouvrement des cartes de contact pour deux protéine A et B.

2.2 Transposition en un problème de clique maximum

Le problème de recouvrement maximum peut être transposé en un problème (NP-difficile) de recherche de clique maximum dans un graphe. En théorie des graphes, une clique est un sous-ensemble de sommets deux à deux adjacents. Une clique maximum d'un graphe est une clique comportant le plus grand nombre de sommets. Pour décrire cette transposition, nous nous appuyons sur l'article de Dawn M. Strickland [13], qui propose de définir un graphe spécifique, formé à partir des cartes de contact et dont chaque sommet représente un recouvrement faisable de deux contacts. Une clique maximum donne la maximisation du recouvrement faisable des cartes de contact.

Pour construire ce graphe, on utilise l'ensemble des contacts C^{a} et C^{b} de deux protéines A et B que l'on souhaite comparer. On suppose que dans l'un et l'autre, les contacts sont triés dans l'ordre alphabétique. Pour chaque contact dans C^{a}, on crée une ligne de sommets, et pour chaque contact dans C^{b}, une colonne de sommets. De cette manière on forme une grille de sommets $NP = C^{\mathrm{a}} \times C^{\mathrm{b}}$, oú chaque sommet correspond à un recouvrement d'un contact (A_i, A_j) dans la protéine A avec un contact $(B_{i'}, B_{j'})$ dans la protéine B, induisant l'alignement de A_i à $B_{i'}$ et A_j à $B_{j'}$. Une arête est formée entre deux sommets du graphe si et seulement si les quatre alignements représentés par ces sommets sont tous faisables simultanément. La Figure 4 reprend l'exemple de la Figure 3 et illustre le graphe obtenu avec les cartes de contact de ces deux protéines fictives.

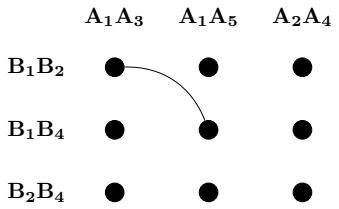


FIGURE 4 – Grille représentant un recouvrement possible.

Une clique dans ce graphe induit un alignement faisable des acides aminés qui n'est pas nécessairement optimal. Une clique maximum donne un alignement optimal. Ce graphe spécifique permet donc de transposer le problème de maximisation du recouvrement des cartes de contact en problème de recherche de clique maximum. Grâce à cela, nous serons en mesure d'utiliser l'algorithme LMC [8].

3 Procédures et perspectives

Notre objectif est de mettre en œuvre cette méthode afin de transformer les instances des benchmarks de la littérature concernant les structures de protéines en instances pouvant être traitées par le solveur LMC. Nous prévoyons dans un premier temps de tester la conversion des instances et leur résolution sur le même ensemble de protéines que celui utilisé dans l'article de Dawn M. Strickland [13]. Par la suite, et en fonction des résultats que nous obtiendrons, nous testerons des ensembles de protéines plus larges.

La première étape consiste à analyser les fichiers représentant la structure tri-dimensionnelle des protéines. Ces fichiers proviennent de la Protein Data Bank (PDB) [2], une très large base de données. Ces fichiers PDB contiennent les coordonnées en trois dimensions des différents atomes constituant les acides aminés de la protéine. En fonction d'un seuil et de la distance euclidienne entre les carbones alpha des

acides aminés, nous formons une matrice d'entiers représentant les contacts entre acides aminés. Lorsque la distance euclidienne entre les carbones alpha de deux acides aminés est inférieure à ce seuil, ils sont considérés comme étant en contact. Une fois achevée, cette matrice représente une carte des contacts de la protéine. Dans un deuxième temps à partir de deux matrices représentant deux protéines différentes, est créé le graphe synthétisant les alignements deux à deux compatibles, suivant la méthode de Dawn M. Strickland. À partir de ce graphe, nous utiliserons l'algorithme LMC afin d'en déterminer la clique maximum.

L'algorithme LMC, abréviation pour Large Max Clique, est très performant sur les graphes constitués d'un grand nombre de sommets avec une faible densité d'arêtes et semble donc particulièrement intéressant pour traiter les instances de ce problème. C'est un algorithme suivant un schéma branch-and-bound dont l'efficacité repose sur une procédure de pré-traitement qui élimine les sommets ne pouvant faire partis de la solution et un raisonnement MaxSAT incrémental permettant de minimiser le nombre de branches dans l'espace de recherche.

Le pré-traitement utilise la notion de noyau. Prenons un graphe non orienté $G = (V, E)$. Un noyau d'ordre k est un sous-graphe G' dont chaque sommet a un degré supérieur ou égal à k. L'ordre d'un sommet v, noté $k(v)$, est l'ordre le plus haut d'un noyau contenant v. Un sommet ne peut appartenir à une clique de taille plus grande que $k(v) + 1$. La procédure trie les sommets de G par degré croissant, calcule l'ordre de chaque sommet, puis réduit ce graphe en éliminant les sommets dont l'ordre est inférieur à la taille de la plus grande clique déjà trouvée de G, permettant de réduire le nombre de sommets, de dériver une grande clique initiale et d'obtenir un ordre initial des sommets.

LMC minimise le nombre de branches en partitionnant G en deux sous-graphes : G_1 et G_2, qui sont initialement vides, de façon à ce que la taille de la clique maximum de G_1 ne dépasse pas la taille lb de la plus grande clique déjà trouvée de G. Pour cela, LMC teste pour chaque sommet v de G suivant l'ordre initial des sommets si la taille de la clique maximum de G_1 avec v est supérieure à lb, en utilisant un raisonnement MaxSAT incrémental. Si la réponse est oui, v est ajouté à G_2, sinon, v est ajouté à G_1. On dénote l'ensemble des sommets de G_1 (G_2) par A et B et le sous-graphe de G induit par A par $G[A]$. Notons $B = \{b_1, b_2, \ldots, b_k\}$. LMC branche successivement en b_i ($i = k, k-1, \ldots, 1$), pour chercher une clique maximum contenant b_i dans le sous-graphe $G[\{b_k, \ldots, b_i\} \cup A]$.

Lorsque que nous obtiendrons les résultats du solveur LMC, nous les étudierons et les comparerons à ceux de la littérature. En fonction de ces résultats, nous souhaitons également adapter le solveur LMC pour qu'il traite de façon plus spécifique ces problèmes, en tirant avantage par exemple des propriétés du graphe obtenu par la méthode de Dawn M. Strickland.

Références

[1] Rumen Andonov, Nicola Yanev, and Noel Malod-Dognin. An efficient lagrangian relaxation for the contact map overlap problem. *International Journal of Computer Applications*, 2008.

[2] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. 2000.

[3] Alberto Caprara, Robert Carr, Sorin Istrail, Giuseppe Lancia, and Brian Walenz. 1001 optimal PDB structure alignments : Integer programming methods for finding the maximum contact map overlap. 2004.

[4] Gergely Csaba, Fabian Birzele, and Ralf Zimmer. Systematic comparison of SCOP and CATH : a new gold standard for protein structure analysis. *BMC Structural Biology*, 9(23), 2009.

[5] Eleanor J. Gardiner, Peter J. Artymiuk, and Peter Willett. Clique-detection algorithms for matching three-dimensional molecular structures. *International Journal on Emerging Technologies*, 1997.

[6] A. Godzik, J. Skolnick, and A. Kolinski. Regularities in interaction patterns of globular proteins. *Protein Engrg*, 1993.

[7] Deborah Goldman, Sorin Istrail, and Christos H. Papadimitriou. Algorithmic aspects of protein structure similiraty. *International Journal of Computer Applications*, 1999.

[8] Hua Jiang, Chu Min Li, and Felip Manyà. Combining efficient preprocessing and incremental maxsat reasoning for maxclique in large graphs. In *In Proceedings of 22nd European Conference On Artificial Intelligence (ECAI-2016), pages 939-947*, 2016.

[9] W. Kabsch. A discussion of the solution for the best rotation to relate two sets of vectors. 1978.

[10] Noel Malod-Dognin. *Protein Structure Comparison : From Contact Map Overlap Maximisation to Distance-based Alignment Search Tool*. PhD thesis, Universite de Rennes, 2010.

[11] Noel Malod-Dognin, Rumen Andonov, and Nicola Yanev. Maximum cliques in protein structure comparison. 2015.

[12] Noel Malod-Dognin and Natasa Przulj. GR-align : fast and flexible alignment of protein 3d structures using graphlet degree similarity. 2014.

[13] Dawn M. Strickland, Earl Barnes, and Joel S. Sokol. Optimal protein structure alignment using maximum cliques. *Operations Research*, 53(3), 2005.

Langage pour la vérification de modèles par contraintes

Pierre Talbot* **Clément Poncelet**

Institut de Recherche et Coordination Acoustique/Musique (IRCAM)
Université Pierre et Marie Curie, Paris, France
{talbot,poncelet}@ircam.fr

Résumé

La vérification de modèles consiste à établir la satisfiabilité d'une formule logique pour un système donné. Ceci implique l'exploration des états du système par des algorithmes de recherche exhaustifs pouvant être couteux. Ces algorithmes sont souvent propres aux outils de vérification de modèles et, à l'instar de la programmation par contraintes, efficaces seulement pour une classe de problèmes. Ces outils manquent donc de modularité. Afin de résoudre ce problème, nous proposons le paradigme de programmation espace-temps, basé sur la théorie des treillis et la programmation synchrone, qui considère une stratégie de recherche comme un ensemble de processus collaborant pour explorer un espace d'état. Nous utilisons ce langage pour réunir la programmation par contraintes et la vérification de modèles et explorons son utilité pour vérifier des propriétés logiques.

Abstract

Model-checking is a paradigm for verifying logic properties on a given system by exploring exhaustively the state-space of the system. However, and similarly to constraint satisfaction problems, there is not a single algorithm working well for all problems. The existing tools lack of modularity and compositionality. To solve these problems, we propose spacetime programming, a paradigm based on lattices and synchronous process calculi that views search strategies as processes working collaboratively towards the resolution of a CSP. Using this language, we investigate relations between model-checking and constraint programming for verifying properties on a model.

1 Introduction

La vérification de modèles [1]—*model-checking* en anglais—est une technique qui vise à établir la satisfiabilité de propriétés logiques pour un modèle donné. De nombreux vérificateurs de modèles ont été implémentés et utilisés en industrie pour vérifier des systèmes imposants [19, 2, 11]. Malgré cela, le problème de l'*explosion d'espace d'état* est fréquemment rencontré et la vérification d'un modèle devient trop couteuse (en temps ou en espace mémoire).

Pour palier à ce problème, plusieurs techniques et paradigmes sont utilisés dont la programmation par contraintes [6, 7, 16]. Les approches existantes proposent des algorithmes efficaces pour une classe spécifique de problème et leurs extensions restent difficiles. De plus, la combinaison du paradigme par contraintes avec la vérification de modèle est enchevêtrée et difficile à comprendre.

Nous proposons d'utiliser spacetime, un paradigme de programmation basé sur la théorie des treillis et la programmation synchrone (section 3). Un programme spacetime considère des stratégies de recherche comme des processus qui explorent simultanément un espace d'état. On compose ensuite ces stratégies à l'aide d'un temps logique, propre au paradigme synchrone, qui synchronise les processus. De cette façon, nous utilisons la programmation par contraintes pour détecter les états inatteignables d'un modèle (section 4). Dès lors, l'idée principale est de composer la stratégie de recherche permettant de montrer la satisfiabilité d'un problème de contraintes avec celle permettant de vérifier des propriétés sur les états du système.

Ces travaux sont préliminaires et le système présenté dans cet article reste en cours d'implémentation. Nous avons néanmoins un compilateur d'un fragment du langage présenté qui permet de spécifier des stratégies d'exploration [1].

*Papier doctorant : Pierre Talbot est auteur principal.

1. Disponible publiquement sur github.com/ptal/bonsai.

2 Préliminaires

2.1 Vérification de modèles

La vérification de modèles [1] consiste à établir la satisfiabilité d'une formule logique Φ pour un modèle $\mathcal{M}$ donné. Ce problème suppose deux modélisations : celle du système (sous la forme d'une machine à état fini) et de la propriété à satisfaire (sous forme d'une formule logique). Une formule Φ est composée de plusieurs *formules atomiques* définissant l'ensemble AP qui est soit une étiquette du modèle—signifiant "mon état est dans ce nœud"—ou une contrainte sur des valeurs de variables (e.g. $x > 0$). Une méthode d'étiquetage (couramment dénotée $\mathcal{L}$) retourne pour un état s de $\mathcal{M}$ l'ensemble des *formules atomiques* satisfaites par s. Le modèle $\mathcal{M}$ est ensuite parcouru pour vérifier, si pour chaque état s, l'ensemble retourné par $L(s)$ satisfait la formule Φ. Finalement, le résultat d'une vérification de modèles est soit la confirmation de la satisfaction de Φ par le modèle $\mathcal{M}$, soit son infirmation accompagnée d'un contre-exemple guidant l'utilisateur depuis un état initial de $\mathcal{M}$ jusqu'à un état atteignable s_{erreur} invalidant la propriété Φ.

Les algorithmes de parcours de $\mathcal{M}$ sont au centre de la complexité des vérificateurs de modèle qui pour les plus simples sont une extension des parcours en profondeur ou en largeur d'un graphe. En revanche, les plus compliqués (par exemple [4]) visent à combiner différents algorithmes d'exploration pour optimiser le temps et l'espace nécessaire lors du calcul de la satisfiabilité de Φ.

Formellement, on définit un modèle $\mathcal{M}$ comme un ensemble de *graphes de programme* (modélisant le graphe de flux de contrôle -CFG- et les synchronisations de chaque programme) : $PG_i = \langle Loc, \mathsf{Effect}, \hookrightarrow, Loc_0, g_0 \rangle_i$ sur un même ensemble d'actions Act_τ, d'instructions $Stmt$ et de variables Var, où :

(i) Loc est un ensemble de nœuds (en anglais *locations*) ;

(ii) $\mathsf{Effect} : Stmt \times \mathsf{Eval}(Var) \to \mathsf{Eval}(Var)$ est un ensemble de fonctions d'effets ;

(iii) $\hookrightarrow$: $Loc \times \mathsf{Cond}(Var) \times Act \times Stmt \times Loc$ est l'ensemble de transitions ;

(iv) $Loc_0 \subseteq Loc$ est un ensemble des nœuds initiaux ;

(v) $g_0 \in \mathsf{Cond}(Var)$ est un ensemble de conditions initiales sur les variables de PG_i.

Avec $\mathsf{Cond}(Var)$ l'ensemble des conditions booléennes sur l'ensemble Var et $\mathsf{Eval}(Var)$ la fonction d'évaluation de variables. On notera $\ell \xrightarrow{g:a:\alpha} \ell'$ pour $\langle \ell, g, a, \alpha, \ell' \rangle \in \hookrightarrow$ avec ℓ le nœud source et ℓ' le nœud cible d'une transition, g sa garde (la condition pour la prise de cette transition) et a (resp. α) l'action de communication (resp. l'effet) de celle-ci.

La sémantique d'un ensemble de graphes de programme est définie par un système de transition TS qui décrit le comportement de n programmes $PG_1 \| \ldots \| PG_n$ s'exécutant en parallèle. TS est défini par $\langle \vec{S}, Act, \to, \vec{I}, AP, L \rangle$ où : (i) $\vec{S} = \langle s_1, \ldots, s_n \rangle$ avec $s_i : Loc_i \times \mathsf{Eval}(Var)$ est une paire constituée d'un nœud du programme PG_i et des valeurs de ses variables ; (ii) $Act = \bigcup_{i \in [1 \ldots n]} Act_i \cup \tau$ est l'ensemble des actions des programmes avec l'action τ spécifiant les transitions internes d'un système ; (iii) $\to$: $\vec{S} \times Act \times \vec{S}$ sont les transitions possibles définies ci-après ; (iv) $\vec{I} = \langle I_1, \ldots, I_n \rangle$ le vecteur d'états initiaux, (v) $AP = \bigcup AP_i$ est l'ensemble des propositions atomiques, et (vi) $L(\langle \ell, \eta \rangle) = \{\ell\} \cup \{g \in \mathsf{Cond}(Var) \mid \eta \models g\}$ est la fonction d'étiquetage retournant les propositions atomiques satisfaites pour un état $s = \langle \ell, \eta \rangle \in \vec{S}$. Les transitions $\to$ sont définies par les règles suivantes :

$$\frac{s_i = \langle \ell_i, \eta_i \rangle \xrightarrow{g:\tau:\alpha}_i \langle \ell_i', \eta_i' \rangle = s_i' \qquad \eta_i \models g}{\langle s_1, \ldots, s_i, \ldots, s_n \rangle \xrightarrow{\tau} \langle s_1, \ldots, s_i', \ldots, s_n \rangle}$$

avec $\eta_i' = \mathsf{Effect}(\alpha, \eta_i)$

$$\frac{\begin{array}{cc} s_i = \langle \ell_i, \eta_i \rangle \xrightarrow{g_i:a?:\alpha_i}_i \langle \ell_i', \eta_i' \rangle = s_i' & \eta_i \models g_i \\ s_j = \langle \ell_j, \eta_j \rangle \xrightarrow{g_j:a!:\alpha_j}_j \langle \ell_j', \eta_j' \rangle = s_j' & \eta_j \models g_j \end{array}}{\langle s_1, \ldots, s_i, \ldots, s_j, \ldots, s_n \rangle \xrightarrow{a} \langle s_1, \ldots, s_i', \ldots, s_j', \ldots, s_n \rangle}$$

avec $\eta_k' = \mathsf{Effect}(\alpha_k, \eta_k)$ pour $k = i, j$ et $a \in Act$. Nous définissons la fonction $\mathsf{post}(\vec{S}) = \{\ell \xrightarrow{g:a:\alpha} \ell' | s = \langle \ell, \eta \rangle \in \vec{S} \wedge \langle \ell, \eta, g, a, \alpha, \ell', \eta' \rangle \in \to\}$ qui pour un ensemble d'états $\vec{S}$ retourne toutes les transitions actives des graphes de programmes sous-jacents.

Pour représenter nos propriétés à vérifier, nous considérons dans ce papier un sous-ensemble de la logique du temps arborescent—plus connue sous le nom de *computation tree logic* (CTL) [3]. Une formule CTL se décompose en deux parties : les propriétés sur les états, notées Φ, et celles sur les chemins, notées φ. Nous utilisons la syntaxe classique pour les états et le chemins sur l'ensemble des propositions atomiques AP :

$$\Phi ::= \text{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

$$\varphi ::= \bigcirc\, \Phi \mid \Phi_1 \,\mathsf{U}\, \Phi_2$$

avec $a \in AP$.

Pour terminer, nous appelons Sat l'ensemble des états satisfaisant une formule CTL Φ. Un modèle satisfait une propriété Φ si $\exists i \in I \mid i \models \Phi$, c'est-à-dire si au moins un de ses états initiaux satisfait la formule Φ.

2.2 Programmation par contraintes

La programmation par contraintes est un paradigme déclaratif constitué de variables, définies sur des domaines, et des relations entre ces variables—appelées contraintes. On définit un problème de satisfaction de contraintes—*Constraint Satisfaction Problem* (CSP) en anglais—comme un tuple $\langle d, C\rangle$ où d est une fonction des variables vers un ensemble de valeurs, appelé le *domaine* de la variable, et C est l'ensemble des contraintes posées sur les variables. En pratique, un CSP est résolu en entrelaçant deux étapes : la propagation qui enlève les valeurs des domaines qui ne satisfont pas au moins une contrainte, et une étape de recherche qui permet de faire un choix lorsqu'on est "bloqué" et de revenir en arrière si le choix était mauvais. Dans cette article, on considère les algorithmes de recherche exhaustifs sur des domaines finis. Nous donnons quelques définitions mathématiques telles que trouvées dans [22].

Formellement, on considère un ensemble de variables $\mathcal{X}$ et un ensemble de valeurs $\mathcal{V}$. Une affectation d'une variable à une valeur est une fonction $a : \mathcal{X} \to \mathcal{V}$. On note l'ensemble de toutes les affectations avec l'exponentiation ensembliste $Asn := \mathcal{V}^{\mathcal{X}}$. Une contrainte $c \in 2^{Asn}$ est l'ensemble de toutes les valeurs acceptées par cette contrainte. Une affectation $a \in c$ est une solution de la contrainte c. Sans perdre en généralité, on considère qu'une variable est toujours contrainte par un domaine et on définit la fonction d tel que $d : \mathcal{X} \to Asn$. Par conséquent, un domaine est un ensemble d'affectations et est défini par la contrainte $con(d) := \{a \in Asn \mid \forall x \in \mathcal{X} : a(x) \in d(x)\}$. Les solutions d'un CSP $\langle d, C\rangle$ sont définies par $sol(\langle d, \mathcal{C}\rangle) := \{a \in con(d) \mid \forall c \in \mathcal{C} : a \in c\}$.

Opérationnellement, les contraintes sont implémentées par des algorithmes de filtrage—appelés propagateurs—réduisant le domaine des variables. Soit l'ensemble des domaines $Dom := 2^{\mathcal{V}^{\mathcal{X}}}$, un propagateur p est une fonction $p : Dom \to Dom$ qui doit vérifier deux propriétés :

(P1) Contractant : $\forall d \in Dom, p(d) \subseteq d$ ce qui signifie que p peut seulement réduire les domaines.

(P2) Correct : $\forall d \in Dom$ et $\forall a \in d$, $p(\{a\}) = \{a\}$ implique $a \in p(d)$. Le propagateur p ne peut pas rejeter des affectations valides.

Le rôle d'un propagateur est donc de réduire les domaines mais aussi de vérifier si une affectation est valide. Un propagateur $p \in P$, où P est l'ensemble des propagateurs, induit une contrainte c tel que $c_p := \{a \in Asn \mid p(\{a\}) = \{a\}\}$. Un problème de propagation $\langle d, P\rangle$ est un CSP $\langle d, \{c_p \in \mathcal{C} \mid p \in P\}\rangle$.

Soit un problème de propagation $\langle d, P\rangle$ et $p_1, \ldots, p_n \in P$, l'étape de propagation consiste à obtenir un point fixe où $d' \subseteq d, p_1(\ldots p_n(d')) = d'$, c'est-à-dire que les propagateurs ne peuvent plus réduire les domaines. La théorie de la propagation étudie des algorithmes permettant d'obtenir ce point fixe le plus rapidement possible [22].

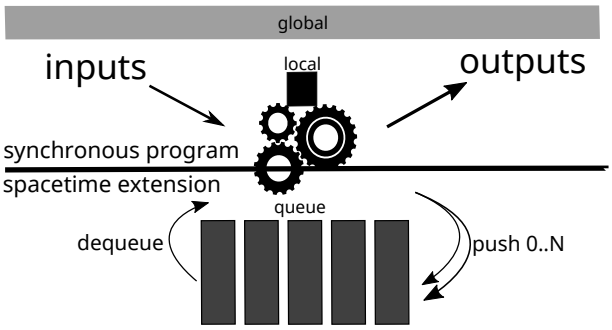


FIGURE 1 – Extension espace-temps du paradigme synchrone.

L'étape de propagation ne permet pas de résoudre un problème de contraintes. En effet considérons $x \neq y$ avec $x, y = \{1, 2\}$, le propagateur $p_{\neq}$ ne peut pas réduire davantage les domaines sans violer $P2$. Par conséquent on doit faire un choix afin d'énumérer les solutions possibles. La technique usuelle est d'utiliser une fonction $branch : Dom \to 2^{Dom}$ qui divise les domaines tel que $\forall d.sol(d) = \bigcup_{d' \in branch(d)} sol(d')$.

3 Programmation espace-temps

3.1 Définitions

Un ensemble ordonné $\langle D, \leq\rangle$ est un treillis complet si chaque sous-ensemble $S \subseteq D$ a une plus petite borne supérieure et une plus grande borne inférieure. Un treillis complet est toujours borné, c'est-à-dire qu'il possède un supremum $\top \in D$ tel que $\forall x \in D.x \leq \top$ et un infimum $\bot \in D$ tel que $\forall x \in D.\bot \leq x$.

Dans cet article, on considère la relation de déduction $x \models y \equiv y \leq x$ pour $x, y \in D$ qui signifie que y peut être déduit de x. On parle aussi de l'opération de jointure $x \vee y$ qui est égale à la plus petite borne supérieure de l'ensemble $\{x, y\}$ dans l'ensemble D. On note $x \leftarrow y$ pour $x = x \vee y$.

3.2 Paradigme de programmation

La programmation espace-temps—dont nous appellerons le langage spacetime—est un paradigme qui étend le modèle synchrone [8] avec des variables définies sur des treillis et un opérateur de non-déterminisme (dans le sens du *backtracking* ici). L'exécution d'un programme synchrone est décomposée en

une séquence d'instants logiques conceptuellement instantanés. Le programme produit donc des valeurs à l'instant t en fonction de l'instant $t-1$ et de signaux de l'environnement. Le paradigme synchrone permet de mettre en parallèle des processus de manière déterministe qui avancent simultanément d'instant en instant.

En spacetime, on voit un processus comme une stratégie d'exploration d'un espace d'état et un programme comme un ensemble de processus évoluant de manière synchrone. De plus, nous utilisons des variables avec une structure de treillis ce qui permet aux processus d'interagir dans l'instant sans que cela pose les problèmes habituels de *deadlock* ou d'indéterminisme. En effet, lorsque deux processus écrivent dans une variable en même temps, la valeur écrite est automatiquement combinée grâce à l'opérateur de jointure du treillis [21]. Ces treillis sont programmés dans un langage hôte, Java dans notre cas, sous forme de classes et nous imposons que toutes méthodes sur ces treillis soient monotones : on ne peut que rajouter de l'information. Cette idée de méthodes monotones sur des objets avec une structure de treillis nous vient du langage Bloom^L [5] utilisé pour programmer des systèmes distribués.

Dans la Figure 1, nous voyons dans la partie haute qu'un programme synchrone répond à des stimulus externes et peut stocker des variables dans une mémoire globale—préservant la valeur des variables au travers des instants—ou locale à l'instant courant. Afin de représenter le non-déterminisme, nous utilisons une troisième mémoire, sous forme de file, représentant l'espace d'état restant à explorer. À chaque instant, un nœud est enlevé de cette file et instancié. À la fin de chaque instant, l'opérateur non-déterministe `space` nous permet d'ajouter N éléments dans cette file représentant N nouveaux états successeurs à explorer dans les prochains instants. Lors de la déclaration d'une variable, nous utilisons trois attributs afin de distinguer dans quelle mémoire on stock les variables : (1) `single_time` pour les variables dans la mémoire locale qui sont ré-initialisées à leur valeur d'origine entre chaque instant ; (2) `single_space` pour les variables dans la mémoire globale et (3) `world_line` pour les variables se plaçant dans la file. Pour les variables `single_space` et `world_line`, on impose que leur évolution soit monotone pour l'entièreté ou un chemin de l'exploration de l'espace d'état.

3.3 Un solveur de contraintes minimal

Le paradigme espace-temps est utile pour isoler les différents composants d'un solveur de contraintes dans des processus distincts. On considère un solveur basique mais fonctionnel basé sur la bibliothèque Choco [17]. En particulier, nous considérons les classes `VStore` et `CStore` qui sont des abstractions (sous forme de treillis) des variables et des contraintes utilisées en Choco [2].

Nous utilisons spacetime pour programmer des stratégies d'exploration de l'arbre généré par la résolution d'un problème de contraintes. Afin de pouvoir composer des stratégies, nous utilisons l'interface suivante :

```
interface Search =
  proc search;
end
```

Elle impose aux modules qui vont l'implémenter de spécifier un processus qui s'appelle `search`. On utilise un module principal implémentant cette interface pour composer les différentes sous-stratégies :

```
module Solver implements Search =
  ref world_line VStore domains;
  ref world_line CStore constraints;
  proc search =
    trap FoundSolution in
      par
      || stop_on_solution()
      || propagation()
      || branch()
      end
    end
  [...]
```

Ce module reflète la définition mathématique d'un CSP $\langle d, C\rangle$ au travers des variables `domains` et `constraints`. Le mot-clé `ref` permet de référencer des variables qui sont des paramètres du module. En effet, le modèle du CSP est donné en entrée du module et est défini de manière usuelle à l'aide des méthodes du système de contraintes sous-jacent. Les différents composants de l'algorithme d'exploration sont codés séparément et composés à l'aide de l'instruction parallèle `par`. Cet opérateur parallèle est compilé vers du code séquentiel et n'est donc pas un parallèle au sens du *multithreading*. Tous les processus utilisent la boucle `loop` P qui exécute le programme P indéfiniment. Afin de respecter la condition qu'un instant doit être "instantané", il faut que P contienne l'instruction `pause`, retardant l'exécution du reste du processus à l'instant suivant.

Le premier processus permet d'arrêter l'exploration lorsqu'on a trouvé une solution :

```
proc stop_on_solution =
  loop
    when domains |= constraints then
      exit FoundSolution;
    end
    pause;
  end
```

2. Le code correspondant est relativement court : 200 lignes de code.

On utilise l'expression conditionnelle `when` afin de lancer l'exception `FoundSolution` lorsqu'on a trouvé une solution. Quand l'exception est lancée, le corps de l'instruction `trap` est terminé dans l'instant suivant et dans ce cas, le programme sera arrêté. L'expression `domains |= constraints` est vraie lorsqu'on peut déduire les contraintes du domaines, c'est-à-dire lorsqu'on a une solution au CSP (on décrit cette relation plus en détail en section 6).

On délègue la propagation des domaines au solveur Choco via le code suivant :

```
proc propagation =
  loop
    constraints.propagate(domains);
    pause;
  end
```

On appelle la méthode `propagate` sur les contraintes qui va se charger de réduire les domaines. Cette opération est bien monotone vu qu'on ne fait que rajouter de l'information dans les domaines.

Finalement, le processus `branch` se charge de couper le domaine des variables :

```
proc branch =
  loop
    single_time IntVar x = fail_first_var (domains);
    single_time Integer v = middle_value(x);
    space
    || constraints <- x.leq(v);
    || constraints <- x.ge(v);
    end
    pause;
  end
end
```

Ce processus décrit l'arbre de recherche avec une stratégie "fail-first". Elle choisit la variable x qui a le plus petit domaine afin de "rater au plus vite" et ne pas s'aventurer trop loin dans l'arbre—ce qui a un coût non négligeable. La fonction `middle_value` permet ensuite de récupérer la valeur v représentant le milieu du domaine. Finalement, l'arbre est construit avec l'instruction `space` qui décrit deux futurs différents : dans l'un $x \leq v$ et dans l'autre $x > v$.

4 Description de l'espace d'état

Nous reprenons les définitions des sections 2.1 et 2.2 pour définir formellement un problème de vérification de modèles par un problème de programmation par contraintes. Pour cela nous utilisons le langage spacetime présenté en section 3 et une abstraction du système de transition ts fournissant les méthodes suivantes :

post retourne l'ensemble des transitions possibles pour l'état courant du système. Le comportement de cette fonction suit la définition de la section 2.1 sans se soucier des valeurs des variables (nous projetons un état du système sur les nœuds courants des programmes uniquement, dénotés $\vec{\ell}$).

apply applique une transition à l'état courant du système. Ceci a pour effet de mettre à jour l'état du système en fonction de la transition.

4.1 Mise en commun des deux formalismes

L'espace d'état d'un problème de vérification de modèles doit être compris en terme d'une structure de treillis. De la sorte, on peut utiliser ce système comme une variable dans un programme spacetime. Les notations de ces deux formalismes ont l'équivalence suivante : la vérification de modèles utilise les ensembles Var, $\mathsf{Eval}(Var)$ dénotés η et Effect qui correspondent respectivement aux ensembles $\mathcal{X}$, à la fonction d'affectation a et à l'ensemble Asn définis en section 2.2.

Une question importante est de transformer le modèle vers un programme spacetime correspondant. On définit le système de transition abstrait ts, de sorte que chaque variable d'un graphe de programme PG soit définie sur un domaine. L'union de ces variables forme donc le premier élément d d'un CSP $\langle d, C\rangle$. Les gardes des transitions peuvent être ajoutées à l'ensemble C sans traitement particulier. En revanche les affectations, par exemple $x := x + 3$, ne peuvent pas être directement transformées vers une contrainte (car la contrainte correspondante est insatisfiable). L'idée est d'utiliser les contraintes de flux [12] et de considérer une variable x comme un flux de valeurs $x_1, \ldots, x_n$ où x_i est la $i$ème affectation de x. De cette manière l'affectation précédente est transformée vers la contrainte $x_{i+1} = x_i + 3$. Ces contraintes de flux permettent de lier les variables du problèmes de contraintes dans des états différents du modèle—nous revenons sur ce point en section 4.2.

L'espace d'état est un treillis de la forme $\langle cs, \vec{\ell}\rangle$ où : $cs : \langle d, C\rangle$ est le système de contraintes courant, et $\vec{\ell} : Loc^n$ est l'ensemble des n nœuds actifs de l'état courant (pour chaque PG_i avec $1 \leq i \leq n$).

L'ordre de ce treillis est défini tel que $\langle cs, \vec{\ell}\rangle \leq \langle cs', \vec{\ell'}\rangle$ si $cs < cs'$ ou si $cs' = cs$ alors $\vec{\ell} \leq \vec{\ell'}$. Ceci nous permet de détecter que l'on repasse sur un nœud déjà visité, c'est-à-dire quand $(\vec{\ell} = \vec{\ell'})$ sans avoir plus d'information $(cs \models cs')$.

On implémente en spacetime le module principale `ModelChecker` :

```
module ModelChecker =
  world_line VStore domains;
  world_line CStore constraints;
  world_line TS ts;
  [...]
```

À l'instar de `Solver` (section 3), ce module contient le CSP courant avec les variables `domains` et `constraints`. Le type de la variable `ts` est la classe Java `TS` fournissant les méthodes `post` et `apply`. On remarque que ces variables sont déclarées avec l'attribut `world_line` ainsi, lors d'un retour en arrière dans l'espace d'état, elles seront automatiquement restaurées.

4.2 Création de l'espace d'état

Une transition sur le treillis traduit "un pas" dans la vérification du modèle. Selon la définition de `post` et avec m la somme du nombre des transitions actives pour chaque graphe de programme, nous avons m transitions. Pour notre problème de contraintes, un pas implique : (i) le calcul des transitions possibles à partir de l'état courant $\vec{\ell}$, (ii) pour chacune de ces transitions $\ell_i \xrightarrow{g:a:\alpha} \ell'_i$:

1. le calcul de l'état cible ℓ', et,
2. l'ajout de la garde g et affectation α dans l'ensemble de contraintes : $C' = C \wedge g \wedge \alpha$.

(iii) le calcul du prochain "pas" pour chaque état cible, avec comme état du treillis $\langle cs', \vec{\ell'} \rangle$.

Remarque : Il est possible que deux transitions soient appliquées (si une communication est présente), dans ce cas notre système de transition retourne l'union des deux gardes ($g = g_1 \wedge g_2$) et des deux affectations ($\alpha = \alpha_1 \wedge \alpha_2$) pour traiter l'étape ii.2.

De ces définitions, on implémente le processus qui définit l'espace d'état :

```
module ModelChecker =
  [...]
  proc model_checker =
    par
    || next_transition  ();
    || prune_unreachable ();
    end
```

Le processus `model_checker` lance deux processus en parallèle : `next_transition` s'occupe de passer les transitions suivantes du système ts, et `prune_unreachable` surveille l'état du système courant et coupe les états non atteignables. Le premier est implémenté comme suit :

```
proc next_transition =
  loop
    single_time Set<Transition> next_transitions =
      ts.post(domains, constraints);
    space t in next_transitions
    || ts.apply(t, domains);
       constraints <- t.guards();
       constraints <- t.effects();
    end;
    pause;
  end
```

Le calcul des prochaines transitions est donc géré par l'appel `post` sur la variable `ts` qui retourne un ensemble de transitions. On visite chacune de ces transitions dans des instants futurs. En programmation par contraintes, il s'agit de la fonction `branch` présentée à la section 3. On utilise la syntaxe `t in next_transitions` pour créer dynamiquement m futurs possibles (et donc passage de transition). On applique les effets de la transition en réalisant la jonction des gardes et des effets de la transition courante avec les contraintes courantes *constraints*. Cette instruction modélise le non-déterminisme présent dans une simulation de modèle lorsque plusieurs transitions sont applicables ; c'est la séparation de l'espace d'état des transitions.

Les contraintes jouent un rôle majeur pour détecter les états inatteignables et vérifier la satisfiabilité du problème de contraintes courant. On sait que résoudre un problème de contraintes nécessite sa propre exploration d'un espace d'état : dans chaque état nous calculons une solution du problème de contraintes courant. On note cependant que le problème de contraintes n'est *pas ré-initialisé* et qu'on repart en fait de la solution précédente, devenue partielle à cause des nouvelles variables de flux générées et contraintes.

```
proc unreachable_prune =
  loop
    universe
      Solver solver = new Solver(domains, constraints);
      solver.solve ();
    end
    when not(domains |= constraints) then
      prune;
    end
    pause;
  end
```

On utilise l'instruction `universe` pour encapsuler la résolution du problème de contraintes. Cette instruction a pour effet de créer une nouvelle file et d'être exécutée tant que la file n'est pas vide, qu'une exception est lancée ou que le programme est terminé. Ce mécanisme est bien connu dans les langages synchrones sous le terme de raffinement temporel [15]. De plus, nous verrons en sections 5 et 6 que cette résolution est combinée avec la vérification de propriété logique. Finalement, si le problème est montré insatisfiable, alors on coupe l'espace d'état courant avec l'instruction `prune`, on ne veut pas explorer de transitions supplémentaires.

5 Formule comme stratégie d'exploration

Nous avons vu en section 2.1 que l'algorithme d'exploration d'un modèle dépend de la formule logique à

vérifier. Les deux logiques les plus répandues sont *Linear Temporal Logic* (LTL) et *Computation Tree Logic* (CTL). Nous nous restreignons à un sous-ensemble de ces logiques pour vérifier des propriétés de sécurité—*safety* en anglais—sur les états du système de transition. La logique considérée ici est la suivante :

$$\phi ::= c \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \forall\Diamond c \mid \exists\Diamond c$$

L'unique atome $c \in C$ est une contrainte définie sur les variables du modèle. On remarque que les quantificateurs ne peuvent pas apparaitre en séquence et nous expliquons cette limitation dans la section 5.1. Afin d'encoder ces formules en spacetime, nous capturons le comportement d'une formule dans l'interface `Formula` qui sera implémentée par les différents opérateurs logiques que nous présentons ensuite :

```
interface Formula =
  Consistency consistency;
  proc satisfy ;
```

On voit un processus comme la spécification de la formule et son implémentation sous forme de stratégie explorant l'espace d'états. Le processus `satisfy` établit la consistance de la formule et stock le résultat dans la variable `consistency`. Le treillis `Consistency` contient les éléments $\{true, false, bot, top\}$ où $bot \leq false \leq top$ et $bot \leq true \leq top$. La valeur `bot` indique donc que la consistance de la formule n'a pas encore été établie, `true` que la formule est satisfiable et `false` insatisfiable.

Encodage de formules logiques La conjonction est encodée par un module prenant en paramètre générique deux autres sous-formules :

```
module Conjunction<F: Formula, G: Formula>
 implements Formula =
  single_time Consistency consistency = bot;
  ref F f;
  ref G g;
  proc satisfy =
    par
    || f.satisfy () <*> g.satisfy()
    || consistency <- f.consistency.and(g.consistency);
    end
```

On exécute en parallèle les deux sous-formules en appelant respectivement leur processus `satisfy`. On remarque que ces deux sous-formules sont composées à l'aide de l'opérateur `<*>` : si une des deux veut couper l'arbre—via `prune`—on doit tout de même attendre la fin de l'autre processus. Par exemple, un processus peut établir que sa formule est vraie avant d'explorer l'entièreté du sous-arbre mais il faut tout de même continuer l'exploration pour le processus parallèle. Le troisième processus met à jour la consistance de la conjonction des formules à l'aide de la méthode `and`. Elle est programmée comme une conjonction classique de deux booléens où `true` $\wedge$ `unknown` est égal à `unknown`. La variable `consistency` est annotée avec `single_time` car la consistance d'une formule n'est valide que dans l'instant courant.

Le module `Negation` permet la négation d'une sous-formule et est implémenté comme suit :

```
module Negation<F: Formula> implements Formula =
  single_time Consistency consistency = bot;
  ref F f;
  proc satisfy =
    par
    || f.satisfy ()
    || consistency <- f.consistency.neg()
    end
```

Similairement à la conjonction on appelle une méthode sur la variable `consistency` de la sous-formule qui, dans ce cas, permet d'inverser la valeur de consistance.

L'encodage des quantificateurs universel et existentiel autorisent la vérification de propriétés dans un chemin. On se concentre ici sur un sous-ensemble où $\forall\Diamond c$ (resp. $\exists\Diamond c$) signifie que tous les (resp. un des) chemins possèdent un état dans lequel l'atome c est vérifié. On encode le quantificateur universel de la manière suivante :

```
module UniversalEventually =
  single_time Consistency consistency = bot;
  ref Atom f;
  proc satisfy =
    trap TreeInconsistent in
      loop
        par
        || f.satisfy ()
        || tree_inconsistency ()
        || path_consistency()
        end
        pause;
      end
    end
  proc tree_inconsistency =
    when f.consistency |= false /\ top then
      consistency <- false;
      exit TreeInconsistent;
     end
  proc path_consistency =
    when f.consistency |= true then
      prune;
    end
```

Ce module ne peut être initialisé qu'avec une formule atomique dû aux restrictions imposées précédemment. Le sous-processus `tree_inconsistency` permet de détecter que l'atome `f` est faux dans la branche courante et que donc, la formule du quantificateur universel est également fausse. La condition `top` est vraie lorsqu'on est dans une feuille de l'arbre (l'espace courant est en échec) et qu'on va revenir en arrière. Dans ce cas, on peut établir l'inconsistance et stopper le processus en lançant l'exception `TreeInconsistent` qui permet de sortir de la boucle. À contrario, le processus

`path_consistency` détecte que le chemin courant est consistent et qu'on peut arrêter l'exploration du sous-arbre courant, d'où l'instruction `prune`. Grâce à notre définition de la conjonction, composant les processus à l'aide de l'opérateur `<*>`, `prune` sera locale à la formule et l'exploration du sous-arbre peut continuer dans un autre processus.

De manière similaire, nous pouvons définir le quantificateur existentiel :

```
module ExistentialEventually<F: Formula> =
  single_time Consistency consistency = bot;
  ref Atom f;
  proc satisfy =
    trap PathConsistent in
      loop
        par
        || f.satisfy()
        || path_consistency()
        end
        pause;
      end
    end
  proc path_consistency =
    when f.consistency |= true then
      consistency <- true;
      prune;
      exit PathConsistent;
    end
```

Dans ce cas, on cherche à trouver un chemin consistent et l'on s'arrête dès que l'atome sous-jacent est satisfiable. On lance également l'exception `PathConsistent` qui permet de sortir de la boucle et arrêter le processus.

On termine par le module `Atom` implémentant une propriété atomique sous forme de système de contraintes :

```
module Atom implements Formula =
  single_time Consistency consistency = bot;
  ref world_line VStore domains;
  ref world_line CStore constraints;
  ref world_line CStore property;
  proc satisfy =
    Entailment ent = new Entailment(model, property);
    par
    || ent.entail()
    || consistency <- ent.consistency
    end
```

Ce module prends trois variables en paramètres : le CSP avec les variables `domains` et `constraints` et la propriété à vérifier `property`. La propriété atomique doit être définie sur les mêmes variables que le modèle. On cherche ensuite à établir si $\langle d, C\rangle \models \mathit{property}$, c'est-à-dire si la propriété atomique peut-être déduite de l'état courant. Ce problème de déduction—*entailment* en anglais—a une complexité potentiellement dans NP (en fonction du système de contraintes) et nécessite donc d'être résolu également avec une stratégie d'exploration. On peut réduire le coût de cette requête de déduction car nous devons de toutes façons prouver la satisfiabilité du CSP courant. Grâce à la sémantique de composition d'espace de spacetime, nous pouvons résoudre toutes les requêtes de déduction (générées par les atomes) ainsi que la satisfiabilité du problème en un unique passage dans l'arbre de recherche.

On a vu en section 3 comment implémenter un solveur de contraintes en spacetime et nous pouvons donc l'utiliser pour ce problème. L'algorithme de déduction implémenté par le module `Entailment` est présenté à la section 6.

5.1 Limitation de spacetime pour LTL et CTL

Nous avons utilisé un fragment restreint des logiques généralement utilisées en vérification de modèles telles que LTL et CTL. Prenons l'exemple des opérateurs "jusqu'à" $\Phi_1 \,\mathsf{U}\, \Phi_2$ et "suivant" $\bigcirc\Phi$ présents dans les deux logiques. Le premier est vrai si Φ_1 est vrai jusqu'à ce que Φ_2 soit vrai. Le deuxième est vrai si dans l'état suivant Φ est vrai. Ces formules agissent donc sur des chemins de l'espace d'état. Considérons la formule $(\bigcirc\Phi_1)\mathsf{U}\Phi_2$. On doit vérifier que Φ_1 est vrai dans le prochain état, le problème étant que dans ce prochain état on devra également vérifier $\Phi_1 \wedge \bigcirc\Phi_1$. Intuitivement, il y a deux manières de traiter le problème :

1. Par redémarrage : on "prends de l'avance" sur l'exploration dans le premier état pour vérifier $\bigcirc\Phi_1$. On explore une partie du sous-arbre (en concordance avec Φ_1) et une fois la satisfiabilité établie ou réfutée, on redémarre l'exploration à partir de l'état courant.
2. Par duplication : la formule est dupliquée de tel sorte que dans le second état on vérifie $\Phi_1 \wedge \bigcirc\Phi_1$.

Pour le moment, le paradigme de programmation spacetime ne permet pas d'exprimer ces deux comportements facilement. Notamment la duplication de code n'est pas permise ce qui rend la deuxième proposition difficile. Une solution serait d'introduire un opérateur de réplication $!P$ équivalent à $P \parallel \mathit{pause}; P \parallel \ldots$ tel que trouvé dans le π-calcul [13]. Ce problème intéressant est laissé à de futurs travaux.

6 Algorithme de déduction

6.1 Définitions

Lors de la résolution d'un CSP $\langle d, P\rangle$, une optimisation bien connue consiste à enlever les propagateurs qui ne peuvent plus apporter d'information. Pour cela, on regarde la relation $\langle d, \emptyset\rangle \models \langle d, p \in P\rangle$ que nous noterons plus simplement $d \models_d p$. D'un point de vue algorithmique la relation peut soit être vraie, fausse

ou ne pas exister :

$$d \preceq_{\models_d} p = \begin{cases} \models_d & \text{if } \forall a \in d, p(\{a\}) = \{a\} \\ \not\models_d & \text{if } \forall a \in d, p(\{a\}) = \{\} \\ \not\preceq_{\models_d} & \text{if } \exists a_1, a_2 \in d \text{ s.t.} \\ & p(\{a_1\}) = \{a_1\} \ \wedge p(\{a_2\}) = \{\} \end{cases}$$

On peut généraliser la relation sur un ensemble de propagateurs $d \models_d P$ ce qui est équivalent $\forall p \in P.d \models_d p$. En se basant sur ces définitions, on peut définir un problème de déduction plus général $\langle d, P\rangle \models \langle d, P'\rangle$ comme suit :

$$\langle d, P\rangle \preceq_{\models} \langle d, P'\rangle = \begin{cases} \models & \text{if } \forall d' \in sol(\langle d, P\rangle), d' \models_d P' \\ \not\models & \text{if } \forall d' \in sol(\langle d, P\rangle), d' \not\models_d P' \\ \not\preceq_{\models} & \text{if } \exists d', d'' \in sol(\langle d, P\rangle), \\ & d' \models_d P' \wedge d'' \not\models_d P' \end{cases}$$

Intuitivement, on peut déduire une information P' d'un CSP $\langle d, P\rangle$ si P' est valide pour toutes les solutions de ce CSP. La non-déduction $\langle d, P\rangle \not\models \langle d, P'\rangle$ signifie qu'on ne pourra jamais déduire P' de $\langle d, P\rangle$ même si on rajoute des contraintes dans P. Finalement, on ne sait pas si on peut ou non déduire P' dans le cas où seulement une partie du CSP permet la déduction. Il manque donc des contraintes pour pouvoir donner une réponse définitive.

6.2 Algorithme

L'algorithme spacetime suivant implémente la définition mathématique de déduction :

```
module Entailment =
  single_time Consistency consistency = bot;
  single_space Consistency entailment = bot;
  ref world_line VStore domains;
  ref world_line CStore constraints;
  ref world_line CStore property;
  proc entail =
    trap Contradiction in
      universe
        loop
          par
          || entailment_on_solution()
          || stop_on_contradiction()
          pause;
        end
      end;
      consistency <- entailment
    end
  proc entailment_on_solution =
    when domains |= constraints then
      entailment <- domains |= property;
    end
  [...]
```

De même que l'algorithme de satisfiabilité décrit en section 4, la requête de déduction est locale à un état du système de transition et il faut donc encapsuler cette résolution dans un univers sous-jacent. Dans le sous-processus `entailment_on_solution`, à chaque fois qu'on est dans un nœud solution, on met à jour le statut de la déduction avec la variable `entailment`. Si `entailment` est égale à *true* ou $false$ à la fin, c'est qu'on a pu établir la (non-)validité de la propriété. Dans le cas où cette variable est égale à *top*, c'est que le résultat ne peut pas encore être établi. Dans ce cas, on fait une petite optimisation en arrêtant la vérification de la propriété courante :

```
proc stop_on_contradiction =
  when entailment |= top then
    prune;
    exit Contradiction;
  end
```

Il existe d'autres optimisations notamment en prenant en compte que la vérification d'une propriété peut se faire sur des nœuds non terminaux dans l'arbre d'exploration. Nous ne couvrons pas ces optimisations dans cette article et laissons ce travail pour des travaux futurs.

7 Travaux connexes

La vérification de modèles et la programmation par contraintes ont souvent été utilisé ensemble, notamment pour transformer la vérification de modèle dans un CSP. Principalement, de nombreux travaux utilisent la programmation logique par contraintes (CLP). On peut également encoder un problème de vérification de modèles en CLP et ainsi vérifier des propriétés de sûreté et de vivacité [7, 6, 16]. En particulier, la sémantique de la logique CTL est encodée sous forme de clauses de contraintes et le modèle est abstrait dans un tableau pour réduire le calcul [14].

Ces travaux sont généralement proposés pour répondre à un problème venant d'une application spécifique. En outre, la plupart des solutions existantes transforment un problème de vérification de modèle vers un problème de satisfiabilité de contraintes. Elles permettent difficilement d'utiliser des idées d'algorithmes d'exploration venant des deux mondes.

Dans un autre registre, considérer un programme synchrone comme une formule logique a déjà été envisagé pour tester des programmes dans le même paradigme [20]. Un des avantages de cette approche étant de rendre la formule programmée plus lisible que dans une logique comme CTL. Cependant, cette solution est spécialisée aux systèmes synchrones et ne permet pas de programmer l'exploration d'un espace d'état.

8 Conclusion

Nous avons introduit formellement le domaine de la vérification de modèle et de la programmation par contraintes. Nous proposons d'utiliser le paradigme *espace-temps* afin d'explorer des pistes communes aux algorithmes d'exploration utilisés par les deux domaines. Ceci nous permet de spécifier un algorithme de vérification de modèles utilisant la programmation par contraintes pour vérifier les états inatteignables d'un modèle. De plus, nous considérons la vérification du système à partir d'une formule logique (basée sur un sous-ensemble de CTL). L'idée est de voir cette formule logique comme une stratégie d'exploration dans l'espace d'état. Grâce à spacetime, nous pouvons composer ces formules logiques avec l'algorithme de satisfiabilité d'un problème de contrainte. Ainsi, les processus spacetime décrivent à la fois le parcours dans le système de transitions—en appliquant les gardes et affectations des transitions au CSP courant—et la stratégie de recherche pour résoudre la satisfiabilité d'un sous-ensemble de formules CTL.

Les travaux proposés dans cet article sont préliminaires. Il va sans dire que les travaux futurs sont nombreux, le premier étant de terminer l'implémentation d'un vérificateur de modèles tel que décrit dans cet article. Le compilateur de spacetime permet de programmer des stratégies d'exploration et l'implémentation du vérificateur de modèles basé sur ces stratégies est en cours de développement.

Une piste de recherche intéressante est de considérer les algorithmes d'exploration utilisés en vérification de modèles. Par exemple, la stratégie de recherche "à changement de contexte limité" [18, 10] permet d'obtenir de meilleures contre-exemples en considérant en premier lieu les chemins où les processus s'entrelacent le moins. En fait, cette stratégie est une instance de stratégie bien connue de la programmation par contraintes : la recherche en profondeur limitant le nombre d'erreur de l'heuristique—*limited discrepancy search* [9].

Références

[1] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, May 2008.

[2] T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Quantitative verification of implantable cardiac pacemakers. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 263–272, Dec 2012.

[3] Edmund M. Clarke and E. Allen Emerson. *Design and synthesis of synchronization skeletons using branching time temporal logic*, pages 196–215. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[4] Sylvain Conchon, Amit Goel, Sava Krstic, Alain Mebsout, and Fatiha Zaïdi. Invariants for finite instances and beyond. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 61–68, 2013.

[5] Neil Conway, William R. Marczak, Peter Alvaro, Joseph M. Hellerstein, and David Maier. Logic and lattices for distributed programming. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 1. ACM, 2012.

[6] Giorgio Delzanno and Andreas Podelski. *Model Checking in CLP*, pages 223–239. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[7] Giorgio Delzanno and Andreas Podelski. Constraint-based deductive model checking. *International Journal on Software Tools for Technology Transfer*, 3(3) :250–270, 2001.

[8] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems*. 1993.

[9] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *IJCAI (1)*, pages 607–615, 1995.

[10] Gerard J. Holzmann and Mihai Florian. Model checking with bounded context switching. *Formal Aspects of Computing*, 23(3) :365–389, 2011.

[11] Pim Kars. The application of promela and spin in the bos project (abstract). 1996.

[12] Arnaud Lallouet, Yat Chiu Law, Jimmy HM Lee, and Charles FK Siu. Constraint programming on infinite data streams. In *International Joint Conference on Artificial Intelligence*, pages 597–604, 2011.

[13] Robin Milner. *Communicating and mobile systems : the pi-calculus*. Cambridge University Press, 1999.

[14] Ulf Nilsson and Johan Lübcke. Constraint logic programming for local and symbolic model-checking. In *Proceedings of the First International Conference on Computational Logic*, CL '00, pages 384–398, London, UK, UK, 2000. Springer-Verlag.

[15] Cédric Pasteur. Raffinement temporel et exécution parallèle dans un langage synchrone fonctionnel, 2013.

[16] Andreas Podelski. Model checking as constraint solving. In Jens Palsberg, editor, *Static Analysis : 7th International Symposium, SAS 2000, Santa*

Barbara, CA, USA, June 29 - July 1, 2000. Proceedings, pages 22–37. Springer Berlin Heidelberg, 2000. DOI : 10.1007/978-3-540-45099-3_2.

[17] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2015.

[18] Shaz Qadeer and Dinghao Wu. KISS : keep it simple and sequential. *Acm sigplan notices*, 39(6) :14–24, 2004.

[19] Anders P. Ravn, Jiří Srba, and Saleem Vighio. Modelling and verification of web services business activity protocol. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems : Part of the Joint European Conferences on Theory and Practice of Software*, TACAS'11/ETAPS'11, pages 357–371, Berlin, Heidelberg, 2011. Springer-Verlag.

[20] P. Raymond, X. Nicollin, N. Halbwachs, and D. Weber. Automatic testing of reactive systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS '98, pages 200–. IEEE Computer Society, 1998.

[21] Vijay A. Saraswat and Martin Rinard. Concurrent constraint programming. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 232–245. ACM, 1989.

[22] Guido Tack. *Constraint Propagation – Models, Techniques, Implementation*. PhD thesis, Saarland University, 2009.

Améliorer la propagation : l'Importance d'être Inconsistant

Ghiles Ziat[1*] **Marie Pelleau**[1] **Charlotte Truchet**[2] **Antoine Miné**[1]

[1] Sorbonne Universités, UPMC Univ. Paris 06, CNRS, LIP6, Paris, France
[2] Université de Nantes, France
{prénom.nom}@lip6.fr charlotte.truchet@univ-nantes.fr

Résumé

Les méthodes classiques de résolution de problèmes de satisfaction de contraintes alternent généralement deux étapes : la propagation et l'exploration. La propagation réduit les domaines en fonction des contraintes. Elle peut être vue comme une étape de division de l'espace de recherche en deux parties : le sous-espace inconsistant, qui est éliminé du processus de résolution et le sous-espace indéterminé qui contient les solutions du problème. L'étape d'exploration consiste alors à diviser le sous-espace indéterminé en plusieurs sous-espaces où continuer la résolution. Cette étape est implantée dans la plupart des solveurs de contraintes à l'aide d'heuristiques de coupe qui reposent sur les domaines des variables et/ou les contraintes du problème. Cet article introduit une nouvelle étape dans la résolution appellée *élimination*. Elle divise l'espace de recherche en deux sous-espaces : l'espace indéterminé et l'espace consistant. Elle permet de mieux tirer profit des contraintes et ainsi d'obtenir des frontières plus pertinentes pour la résolution. Cette nouvelle étape est basée sur une observation clé : la partie consistante d'un problème est équivalente à la partie inconsistante du problème complémentaire. Nous avons implanté cette méthode au sein du solveur continu AbSolute. Ce solveur mixe des méthodes d'Interpretation Abstraite et de Programmation Par Contraintes, et la technique d'élimination s'y intègre bien. Les premiers résultats expérimentaux montrent des améliorations significatives des performances du processus de résolution.

Abstract

Classical CSP solving methods often alternate two steps : propagation and exploration. Propagation reduces the domains of the variables according to the constraints. It can be seen as a discrimination of the search space in two sub-spaces : the inconsistent one that can be deleted from the solving process, and the undetermined one which may contain the solutions. The exploration step divides the undetermined sub-space into several sub-spaces in which the search continues. This step is usually implemented in solvers by split heuristics relying onto the domains of the variables and/or the constraints of the problem. This article introduces a new step into the solving process called *elimination*. It divides the search space into two sub-spaces : the undetermined one and the consistent one. It allows the solver to benefit more from the constraints, thus obtaining more significative frontiers for the exploration. It is based on a key observation : the consistent part of a problem is equivalent to the inconsistent part of the complementary problem. This new step is implemented in the AbSolute continuous solver. This solver combines methods from Abstract Interpretation and Constraint Programming. Our elimination technique can be easily added in it. Preliminary results show significative improvements of the solving process.

1 Introduction

La résolution de CSP implique génréralement l'alternance de deux étapes : la propagation et l'exploration. La propagation réduit les domaines en fonction des contraintes. L'exploration quant à elle ajoute des hypothèses afin de diviser le problème en plusieurs sous-problèmes plus faciles à résoudre. Dans le continu, la méthode de résolution peut retourner un pavage de l'espace de recherche à l'aide d'éléments simples à manipuler (des boîtes, en général). Ce pavage peut correspondre à une approximation extérieure ou sur-approximation des solutions, comme dans le solveur Ibex [7], ou peut correspondre à une approximation intérieure ou sous-approximation [8].

Que ce soit dans le discret ou dans le continu, il n'existe pas une unique façon d'explorer l'espace de recherche, et différentes heuristiques existent. Ces heu-

(a) Sous-approximation (b) Sur-approximation

FIGURE 1 – Comparaison entre la sous-approximation et la sur-approximation.

ristiques utilisent généralement la taille des domaines et/ou les contraintes afin de déterminer quelle variable instancier ou quel domaine couper. Dans le discret, les heuristiques les plus connues sont *dom* ou *fail-first* [13] choisissant la variable avec le plus petit domaine, et $dom + deg$ [6], $dom\,/\,deg$ [4] et $dom\,/w\,deg$ [5] choisissant, toutes les trois, la variable en fonction de la taille de son domaine et du nombre de contraintes dans lesquelles elle apparaît. Dans le continu, les heuristiques classiques sont *largest first* [19] choisissant le domaine de plus grande taille à couper, *round robin*, où les domaines sont traités successivement, ou encore *maximal smear* [12] basée sur les dérivées des contraintes et choisissant le domaine avec la plus grande pente. Plus récemment, *Mind The Gaps* [1] reprend l'idée de [12, 19] et utilise des consistances partielles pour couper les domaines.

Dans cet article, nous proposons d'ajouter une nouvelle étape à la méthode de résolution continue : l'élimination. Cette phase divise l'espace de recherche en deux sous-espaces : le premier contenant uniquement des solutions, et le second pouvant contenir des solutions. Notre méthode de résolution alterne donc trois étapes, la propagation, l'élimination et l'exploration. Notre nouvelle étape s'inspire des contracteurs utilisés dans Ibex ([7]) pour réaliser une exploration plus intelligente, de façon assez similaire à *Mind The Gaps* [1]. Afin de s'abstraire de la représentation par intervalles nous ajouterons notre nouvelle étape à la méthode de résolution abstraite présentée dans [17]. Notons que cette méthode permet de calculer simultanément une approximation extérieure et intérieure de l'ensemble des solutions.

Cet article est organisé comme suit : nous rappelons dans la section 2 les notions préliminaires d'intrepretation abstraite nécessaires à la compréhension de la méthode de résolution présentée dans [17]. Nous présentons cette méthode dans la section 3. La section 4 introduit l'étape d'élimination. Nous nous intéresserons aux performances dans la section 5. La section 6 présente les travaux futurs et la conclusion.

2 Rappels

L'interprétation abstraite (IntAbs) [9] est une théorie générale de l'approximation de la sémantique des systèmes discrets dynamiques. Sa principale application est l'analyse statique de programmes, basée sur la sémantique. Elle remplace les calculs trop coûteux de la sémantique réelle par des calculs plus simples, bien que moins précis, réalisés dans un domaine abstrait qui écarte les détails non pertinents. De nombreux domaines abstraits ont été proposés, réalisant divers compromis coût/précision et adaptés à différents types d'analyses. En particulier, pour les propriétés numériques, l'IntAbs propose des domaines abstraits numériques [14], qui peuvent représenter et manipuler efficacement des ensembles de points dans un espace vectoriel représentant une sur-approximation des états de programme accessibles. Un élément clé de l'IntAbs est la dérivation systématique d'une analyse abstraite à partir de la sémantique concrète par application d'une opération d'abstraction. Cette abstraction aboutit à une analyse statique qui est correcte par construction.

3 Résolution abstraite

En PPC, les problèmes de satisfaction de contraintes (CSP) sont modélisés à l'aide de triplets $(\mathcal{X}, \mathcal{D}, \mathcal{C})$:

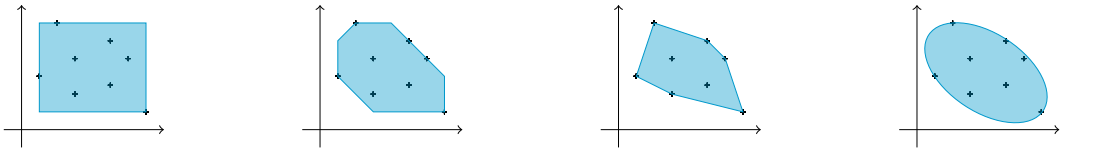
— $\mathcal{X} = \{x_1, ..., x_n\}$, les variables du problème
— $\mathcal{D} = \{d_1, ..., d_n\}$, les domaines de ces variables tels que $x_k \in d_k, \forall k \in [1, n]$
— $\mathcal{C} = \{c_1, ..., c_m\}$, les contraintes du problème

où n est le nombre de variables et m le nombre de contraintes.

Définition 1 (Solution concrète)**.** Une solution d'un CSP est une instanciation de toutes les variables qui respecte toutes les contraintes.

3.1 Un peu d'abstraction

On rappelle ici les notions importantes permettant de définir la méthode de résolution présentée dans [17]. Résoudre un CSP revient à calculer l'ensemble S de ses solutions. Celui-ci pouvant être trop coûteux à calculer



(a) Intervalles (b) Octogones (c) Polyèdres (d) Ellipses

FIGURE 2 – Abstraction d'un ensemble de points avec différents domaines abstraits.

(ou impossible à calculer exactement dans le continu), il peut être plus intéressant d'en trouver une approximation. Nous cherchons ainsi à manipuler un ensemble d'instanciations, représentées par une propriété particulière, plutôt que de les manipuler une à une. Cela se traduit par la construction d'un recouvrement de l'espace de recherche à l'aide d'une disjonction d'éléments abstraits (intervalles, octogones ...).

Définition 2 (Solution abstraite)**.** Soit S l'ensemble des solutions concrètes, on calcule :

- une sur-approximation (ou approximation extérieure) $S^\sharp$ de S telle que : $\forall s \in S \Rightarrow s \in S^\sharp$
- une sous-approximation (ou approximation intérieure) $S^\sharp$ de S telle que : $\forall s \in S^\sharp \Rightarrow s \in S$

Une méthode de résolution est dite *complète* si elle retourne une sur-approximation, et est dite *correcte* si elle retourne une sous-approximation de l'ensemble des solutions. Dans le continu, les méthodes de résolution sont généralement complètes.

La figure 1 compare sur un même exemple le résultat obtenu par sous-approximation (1a) et celui obtenu par sur-approximation (1b). Dans cet exemple on cherche à approximer une zone (en bleu), on peut voir que les boîtes solutions dans la figure 1a sont totalement incluses dans la zone, elles ne contiennent que des solutions. En revanche, dans la figure 1b, certaines boîtes ne contiennent pas que des solutions, on a une sur-approximation de l'espace des solutions.

Dans les deux cas, on peut juger de la qualité de la couverture produite par un solveur selon plusieurs critères. Nous déciderons dans ce travail de se concentrer sur deux critères :

- Le nombre d'éléments : les solveurs étant souvent utilisés comme première étape d'un traitement spécifique, nous essaierons ici de minimiser le nombre d'éléments dans la couverture de l'espace, pour faciliter la réutilisation des résultats du solveur. Les avantages liés à cette réduction peuvent être important dans le cadre d'applications d'approximations intérieures où il est plus intéressant de recouvrir l'espace de peu de "gros" morceaux plutôt que de beaucoup de petits.
- La redondance : si les résultats obtenus ne s'intersectent pas entre eux, alors on dit de la résolution qu'elle est non-redondante. Cette propriété garantit qu'on ne traite pas plusieurs fois les mêmes instanciations concrètes et peut être souhaitable dans le cadre d'applications à la peinture industrielle, (ne pas peindre plusieurs fois une même surface), impression 3D, etc.

La méthode de résolution que nous présentons ici est aussi bien adaptée à une résolution complète que correcte. Elle permet, avec la majorité des domaines abstraits utilisés, d'être non-redondante. Aussi, la technique d'élimination permet sur beaucoup d'exemples de réduire le nombre d'éléments de la couverture.

3.2 Domaines abstraits

La méthode de résolution définie dans [17] se définit en transposant les concepts de propagation et d'exploration aux domaines abstraits. Pour approximer l'espace de solutions, elle utilise des opérations sur un nombre fini d'éléments abstraits. Cette méthode est paramétrée par un domaine abstrait qui redéfinit les concepts usuels de programmation par contraintes.

Les domaines abstraits sont fondés sur la notion de treillis et permettent d'encoder un certain type d'approximation d'états. Un treillis est un ensemble muni d'un ordre partiel (relation réflexive, transitive et antisymétrique). Les domaines abstraits doivent de plus implanter plusieurs opérations nécéssaires à leurs manipulations (union, intersection, élargissement ...). Ils sont très largement utilisés en analyse de programmes [9, 3] et plusieurs domaines, proposant différents compromis coût/précision ont été développés.

Le domaine des intervalles, par exemple, permet d'encoder des relations de la forme : $\bigwedge i \in 1..n,\ a_i < x_i < b_i$ où n est le nombre de dimensions de l'espace, et a_i et b_i sont des constantes. Le domaine des octogones [16] est plus expressif et autorise des relations de la forme $\pm x_i \pm x_j$ entre les variables.

La figure 2 montre l'abstraction d'un même espace avec quatre domaines abstraits parmi les plus utilisés : les intervalles [9], les octogones, les polyèdres [10] et les ellipsoïdes [11].

Nous donnons ici la définition des domaines abstraits tels qu'utilisés dans notre méthode de résolution :

Définition 3 (Domaine abstrait)**.** Un domaine abstrait est défini par :

— un ordre partiel $(E, \subseteq)$, (où les éléments de E sont représentables sur machine),
— une fonction d'abstraction α du domaine des variables vers E,
— une fonction de concretisation γ qui forme une correspondance de Galois avec α,
— un plus petit élément $\bot$,
— un plus grand élément $\top$.

Cette définition classique, fondée sur un ordre partiel, permet d'établir une hiérarchie entre éléments abstraits. Les domaines abstraits que nous utiliserons doivent implanter plusieurs fonctions spécifiques à la résolution de CSP :nous ajoutons à la définition des domaines abstraits les éléments suivants :

Définition 4 (Domaines abstraits pour les contraintes)**.**

— un opérateur de consistance : $cons_E : E^\sharp \to c \to E^\sharp$ (avec c une contrainte du problème),
— un opérateur de coupe $\oplus_E : E^\sharp \to E^\sharp \times \cdots \times E^\sharp$,
— une fonction de taille $\tau_E : E^\sharp \to \mathbb{R}$.

L'ajout d'un opérateur de consistance est nécéssaire pour l'étape de propagation, tandis que l'opérateur de coupe permet de définir une étape d'exploration. Ces opérateurs étant décrits précisément dans [17] et [18], nous ne les décrirons pas dans le détail. Voici cependant un rappel de leur usage. L'opérateur de consistance permet de filtrer des valeurs inconsistantes des éléments abstraits. L'ajout d'une fonction de taille nous permet de définir un critère de terminaison. En effet, l'exploration d'une branche s'arrêtera lorsqu'un élément abstrait de celle-ci est plus petit qu'une certaine valeur fixée à l'avance.

L'opérateur de coupe permet de continuer la recherche dans un élément après l'étape de propagation : lorsqu'un élément ne satisfait pas toutes les contraintes, il est divisé en plusieurs sous-éléments. Nous ne détaillerons pas les différentes heuristiques de coupe d'un élément dans cet article, et utiliseront uniquement la technique de coupe *largest first* qui divise chaque élément en deux, dans un axe perpendiculaire à la plus longue dimension. Ainsi on peut redéfinir une méthode de résolution reprenant les principes de propagation-exploration et basée sur les domaines abstraits. L'algorithme 1 donne le pseudo-code de cette méthode de résolution abstraite.

L'algorithme construit une disjonction d'éléments abstraits. À chaque étape, l'élément courant est filtré par rapport aux contraintes à l'aide de l'opérateur de consistance. S'il ne satisfait pas toutes les contraintes, il est divisé à l'aide de l'opérateur de coupe et on propage les contraintes dans les éléments résultants. On

Algorithme 1 Résolution paramétrée par un domaine abstrait A

```
function SOLVE(D, C, r)        ▷ D : domaines, C : contraintes
    sols ← ∅                    ▷ solutions trouvées
    explore ← ∅                 ▷ éléments à explorer

    e = init(D)                 ▷ initialisation

    push e in explore

    while explore ≠ ∅ do
        e ← pop(explore)
        e ← filtre(e, C)
        if e ≠ ∅ then
            if τ_A(e) ≤ r or satisfait(e, C) then
                sols ← sols ∪ e
            else
                push ⊕_A(e) in explore
            end if
        end if
    end while
end function
```

repète récursivement ces étapes jusqu'à ce que les éléments obtenus satisfassent les contraintes ou soient plus petits qu'un paramètre r de l'algorithme. Si un élément satisfait toutes les contraintes, on l'ajoute directement à l'ensemble des solutions intérieures retourné par l'algorithme. Si un élément ne satisfait pas toutes les contraintes et qu'il est trop petit pour être divisé, il est ajouté à la liste des solutions si l'on désire avoir une résolution complète, ou ignoré si l'on désire avec une résolution correcte.

Les trois fonctions `init`, `filtre` et `satisfait` sont génériques et ne dépendent pas de la représentation du domaine abstrait. La fonction `init` crée un élément abstrait à partir des domaines initiaux du problème. La fonction `filtre` correspond à la boucle de propagation, elle applique la consistance ($cons_A$) pour chacune des contraintes. La fonction `satisfait` quant à elle vérifie si un élément abstrait est une solution pour toutes les contraintes, c'est-à-dire si il ne contient que des solutions. Cette fonction correspond à un contracteur tel que défini dans [7]. Ces fonctions se déduisent facilement de la consistance utilisée et sont données par les pseudos-codes 3, 4 et 5 en annexe.

La figure 3 montre le résultat de l'algorithme 1 pour le CSP suivant :

$$x_1 \in [-4, 10]$$

$$x_2 \in [0, 5]$$

$$x_2 \leq x_1^2$$

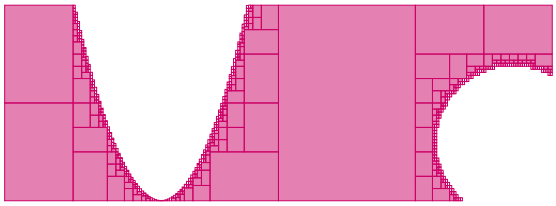


FIGURE 3 – 229 éléments intérieurs, 310 éléments frontaliers.

$$(x_1 - 9)^2 + (x_2 - 1.5)^2 > 4$$

Il est intéressant de noter ici que cette méthode de résolution permet à la fois de sous-approximer l'espace de recherche en tenant compte uniquement des éléments intérieurs, ou de le sur-approximer en tenant compte de tous les éléments retournés. On peut alors s'intéresser à améliorer la couverture produite par le solveur. On peut voir sur la figure 3 que certains des éléments intérieurs pouraient être fusionnés afin d'obtenir de plus grandes boites intérieures. Cette observation traduit le fait que certaines étapes d'exploration sont inutiles. L'étape d'élimination part de cette observation et, en utilisant les contraintes, va essayer de trouver de plus grandes boîtes intérieures.

4 Élimination

L'étape de propagation permet de réduire l'espace de recherche en y retirant des sous-espaces non consistants. Si elle permet alors de sur-approximer l'espace des solutions, on peut définir un symétrique de cette opération pour guider la résolution. Dans cette approche, nous nous intéressons à l'ensemble des instanciations qui ne peuvent être des solutions. Par élimination, le reste de l'espace de recherche ne peut contenir que des solutions. Plus précisément, on cherche à sur-approximer l'ensemble des instanciations qui ne satisfont pas au moins une contrainte. Nous nous référerons par la suite à ces instanciations comme les instanciations inconsistantes ou non-consistantes.

Définition 5 (Complémentaire)**.** Soit $\overline{S}$ le complémentaire des solutions du CSP, c'est-à-dire l'ensemble des instanciations qui ne sont pas solutions.

Cet espace pouvant être incalculable, nous en calculons une sur-approximation $\overline{S}^\sharp$. Cette étape est directe car calculer $\overline{S}^\sharp$ revient à faire une étape de propagation sur la négation des contraintes du CSP.

Nous pouvons alors distinguer dans l'espace de recherche deux sous-espaces :

— l'ensemble des instanciations consistantes inclus dans $S^\sharp$

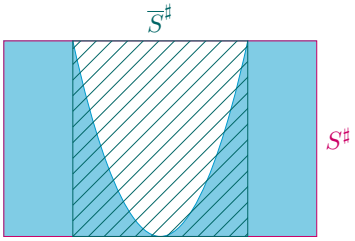


FIGURE 4 – Étant donnée la contrainte en bleu, la boîte $S^\sharp$ sur-approxime les solutions et la boîte hachurée $\overline{S}^\sharp$ sur-approxime les inconsistances.

— l'ensemble des instanciations non-consistantes inclus dans $\overline{S}^\sharp$

La figure 4 donne un exemple du complémentaire pour une contrainte donnée. Selon la contrainte en bleu, la boîte $S^\sharp$ (en rose) sur-approxime les solutions et la boîte $\overline{S}^\sharp$ (hachurée en vert) sur-approxime les inconsistances.

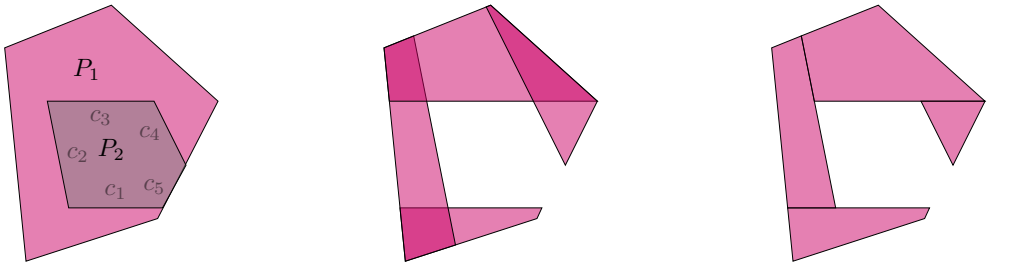
Nous pouvons remarquer que $S^\sharp \setminus \overline{S}^\sharp$ ne contient que des solutions. De plus, $S^\sharp$ peut se définir comme $(S^\sharp \setminus \overline{S}^\sharp) \cup (S^\sharp \cap \overline{S}^\sharp)$. Nous nous baserons sur cette observation pour proposer une amélioration de notre méthode de résolution : nous pouvons directement récupérer de notre espace de recherche les valeurs $(S^\sharp \setminus \overline{S}^\sharp)$, celles-ci étant consistantes par définition, et continuer la résolution dans $(S^\sharp \cap \overline{S}^\sharp)$.

Cette méthode recquiert donc de calculer les espaces $S^\sharp \cap \overline{S}^\sharp$ et $S^\sharp \setminus \overline{S}^\sharp$. Elle n'est donc possible que si les domaines ont un opérateur d'intersection et un opérateur de différence $\setminus$. La plupart des domaines abstraits sont clos par intersection et proposent des opérateurs d'intersection exacts ; calculer $S^\sharp \cap \overline{S}^\sharp$ ne pose aucun problème. Pour calculer $S^\sharp \setminus \overline{S}^\sharp$, il est nécessaire de définir un opérateur de différence.

4.1 Opérateur de différence

Dans cette section, nous cherchons à séparer un élément abstrait e_1 en deux sous-parties : une partie dite consistante, qui satisfait toutes les contraintes du problème et une partie dite non-consistante, qui ne satisfait pas au moins une d'entre elles. Ces deux sous-parties devront recouvrir entièrement e_1. Nous déterminerons ces deux parties à l'aide d'un second élément e_2 qui sur-approxime la partie non-consistante du problème. Nous pouvons alors séparer e_1 en deux parties : $e_1 - e_2$ et $e_1 \cap e_2$. Nous travaillerons ici avec les équivalents abstraits de ces deux valeurs.

Définition 6 (Opérateur de différence)**.** Un opérateur



(a) Polyèdres de départ (b) Différence redondante (c) Différence non redondante

FIGURE 5 – Comparaison du résultat de l'opérateur de différence : $P_1 \ominus P_2$

de différence $\ominus : \mathcal{D}^\sharp \times \mathcal{D}^\sharp \to \wp(\mathcal{D}^\sharp)$ est un opérateur binaire tel que $\forall e_1, e_2 \in \mathcal{D}^\sharp$:

1. $|e_1 \ominus e_2|$ est finie
2. $e_i \in (e_1 \ominus e_2) \Rightarrow e_i \cap e_2 = \emptyset$
3. $\gamma(e_1) = \gamma(e_1 \cap e_2) \cup \{\gamma(e_i) \,|\, e_i \in (e_1 \ominus e_2)\}$

Grâce à cet opérateur, nous pouvons ajouter directement le résultat de $e_1 \ominus e_2$ à l'ensemble des solutions de la résolution, et continuer la recherche dans $e_1 \cap e_2$.

La première condition permet de s'assurer que la résolution produit un ensemble fini de solutions. La deuxième s'assure que l'opérateur est correct, en effet seules les solutions sont traitées comme des solutions. Finalement, la troisième condition garantit la complétude de la résolution, aucune solution n'est perdue.

En pratique, l'utilisation de domaines abstraits convexes classiques (e.g., conjonctions de contraintes linéaires), à même de représenter la négation de contraintes, permet l'implantation d'un opérateur de différence exact.

4.1.1 Pour les domaines convexes

Plusieurs représentations des domaines convexes sont possibles. Dans [17], la représentation par générateurs est utilisée pour calculer les frontières de l'opérateur de coupe tandis que la représentation sous forme de contraintes est préférée pour les calculs de consistance. Nous utiliserons cette seconde représentation dans cette section. Les domaines abstraits les plus populaires (intervalles, octogones, polyèdres) peuvent être définis comme une conjonction de contraintes linéaires. En utilisant cette représentation, l'opérateur de différence est le même pour les polyèdres, les octogones et les intervalles.

Un polyèdre (resp. octogone, intervalle) peut être défini par $\mathcal{P} = \{c_1, ..., c_p\}$, où $c_i : \sum a_i \times x_i \triangleleft b_i$, avec $\triangleleft \in \{<, \leq\}$. Notons ici qu'il est nécessaire de pouvoir exprimer des contraintes strictes et des contraintes larges afin de pouvoir exprimer exactement la négation des contraintes.

On s'intéresse alors à la définition de la différence de deux polyèdres $P_1 \setminus P_2$: on cherche à identifier du polyèdre P_1 ses sous-parties qui n'intersectent pas P_2.

Nous définissons dans un premier temps une version redondante de cet opérateur, i.e., les éléments obtenus peuvent avoir une intersection non vide.

Définition 7 (Différence de polyèdres)**.** Soient deux polyèdres P_1 et P_2 représentés respectivement par les ensembles de contraintes linéaires C_1 et C_2. La différence de P_1 et P_2 s'écrit alors comme :

$$P_1 \ominus P_2 \triangleq \{(P_1 \cap (\neg c)) | c \in C_2\}$$

De façon évidente, $\ominus$ est un opérateur de différence pour les polyèdres qui respecte la définititon 6.

Intuition. $P_1 \ominus P_2$ retourne un ensemble qui à chaque contrainte de P_2 associe un élement. Le nombre de contraintes de P_2 étant fini, l'ensemble est donc fini (6.1). De plus, par définition de l'intersection, la propriété (6.2) est respectée car chaque élément retourné est inclu dans P_1. Enfin, (6.3) est aussi respectée : $P_1 \ominus P_2$ peut s'écrire comme $P_1 \cap \overline{P_2}$. P_1 est entièrement couvert par P_2 et $P_1 \ominus P_2$: aucune solution n'est perdue. Toutefois, cet opérateur présente l'inconvénient de retourner une couverture potentiellement redondante de $P_1 \setminus P_2$. Nous proposons alors une amélioration de cette définition pour obtenir une couverture non redondante.

Définition 8 (Différence de polyèdres non redondante)**.** Soient deux polyèdres P_1 et P_2 représentés respectivement par les ensembles de contraintes linéaires C_1 et C_2, avec $C_2 = \{c_1, \ldots, c_{p_2}\}$. On a alors :

$$P_1 \ominus P_2 \triangleq \{(P_1 \cap c_i) \cap (\neg c_j) \mid i, j \in \{1, \ldots, p_2\}, i < j\}$$

Pour des raisons similaires, les propriétés (6.1), (6.2) et (6.3) de la définition (6) sont aussi respectées. Ici, nous nous assurons de plus que pour toute paire d'éléments $(e_1, e_2) \in P_1 \ominus P_2$, $e_1 \cap e_2 = \emptyset$. Ce résultat est garanti par la considération suivante : chaque élément de

(a) Coupe (b) Élimination puis coupe

FIGURE 6 – Comparaison entre la coupe (figure 6a) et l'élimination suivi d'une coupe (figure 6b)

la disjonction est issu de la négation d'une et une seule contrainte et de la conjonction des contraintes précédentes de P_2. Ainsi, pour tout élément e_i de $P_1 \ominus P_2$, tel que e_i est contraint par la négation de la contrainte c_i, on a : e_i a une intersection vide avec les éléments suivant e_j de $P_1 \ominus P_2$ tels que $j > i$ car ceux-ci sont tous contraints par c_i.

La figure 5 montre un exemple d'application de l'opérateur de différence à deux polyèdres. La figure 5a donne les polyèdres P_1 et P_2 de départ, avec P_2 représenté par les contraintes $\{c_1, \ldots, c_5\}$. La figure 5b montre le résultat de l'opérateur de différence redondant, les redondances apparaissent en rose plus foncé sur la figure. La figure 5c montre le résultat de l'opérateur de différence non redondant et permet de constater qu'il n'existe plus de zones plus foncées.

Ici, $P_1 \ominus P_2$ retourne un ensemble de quatre éléments. P_2 est une conjonction de cinq contraintes. La négation d'une d'entre elles est incompatible avec P_1, elle est ignorée.

Nous avons désormais un nouvel opérateur, avec sa définition théorique et opérationnelle. Nous pouvons désormais définir une nouvelle étape dans la résolution utilisant ce nouvel opérateur.

4.2 Nouvelle étape dans la résolution

Calculer $S^\sharp \setminus \overline{S}^\sharp$ peut permettre de réduire l'espace de recherche. En effet, lorsque l'on ne peut plus filtrer les valeurs non-consistantes du domaine des variables, nous proposons une étape d'élimination avant l'étape d'exploration. Plutôt que de diviser arbitrairement un élément abstrait, l'élimination permet d'identifier les parties ne contenant que des solutions. Elle permet d'ajouter une phase de résolution basée sur les contraintes elle-mêmes et délaie ainsi la phase de coupe qui effectue des choix de division plus arbitraires.

Les figures 6a et 6b comparent les résultats d'une coupe d'un élément et d'une élimination suivie d'une coupe pour une contrainte. On remarque que la frontière de coupe ne tient pas compte de la contrainte dans la figure 6a tandis que dans la figure 6b, les boîtes ne contenant que des solutions sont d'abord conservées (les boîtes totalement roses des deux côtés de la parabole). Puis l'opérateur de coupe, coupe la troisième boîte en deux dans la hauteur.

Algorithme 2 Résolution avec élimination

```
function SOLVE(D, C, r)                    ▷ D : domaines, C : contraintes
    sols ← ∅                               ▷ solutions trouvées
    explore ← ∅                            ▷ éléments à explorer
    e = init(D)                            ▷ initialisation
    push e in explore

    while explore ≠ ∅ do
        e ← pop(explore)
        e ← filtre(e,C)
        if e ≠ ∅ then
            if τ_A(e) ≤ r or satisfait(e,C) then
                sols ← sols ∪ e
            else
                e_non−cons ← complementaire(e,C)
                e_cons ← e ⊖ e_non−cons
                for e_i ∈ e_cons do
                    sols ← sols ∪ e_i
                end for
                push ⊕_A(e ∩ e_non−cons) in explore
            end if
        end if
    end while
end function
```

L'algorithme 2 donne le pseudo-code associé à notre méthode de résolution avec la nouvelle étape d'élimination. La différence avec l'algorithme 1 apparait en bleu. La fonction `complementaire` calcule $e_{non-cons}$, une sur-approximation de l'inconsistance. Puis avec l'opérateur de différence les éléments solutions sont conservés. Finalement, la résolution continue dans l'espace de recherche indéterminé $(e \cap e_{non-cons})$.

La figure 7 montre le résultat de cette méthode sur le CSP donné précédemment. On peut voir que sur cet exemple, on obtient pour la même précision : 100 éléments intérieurs et 273 éléments extérieurs en alternant propagation, élimination et exploration, contre

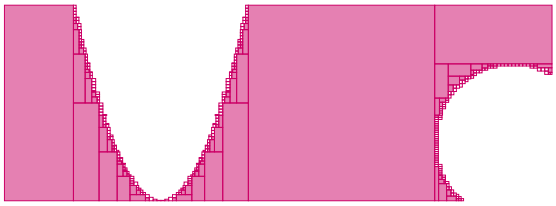


FIGURE 7 – Résolution avec élimination : 100 éléments intérieurs, 273 éléments frontaliers.

223 éléments intérieurs et 310 éléments extérieurs obtenus en alternant propagation et exploration, soit près de deux fois moins d'éléments intérieurs. On analyse dans la section suivante plus finement les performances de notre méthode de résolution.

5 Résultats expérimentaux

Dans cette section, nous expérimentons les performances de la méthode de résolution avec élimination. Nous avons intégré cette technique au sein du solveur AbSolute [1]. Celui-ci implante la méthode présentée initialement dans [17] et la technique d'élimination s'y incorpore particulièrement bien. En effet, celle-ci requiert l'implantation notamment d'une opération de différence ensembliste. Cette opération peut s'avérer coûteuse avec une représentation classique des domaines des variables, tandis que l'utilisation de domaines abstraits permet de réduire son coût.

5.1 Présentation des problèmes

Les problèmes sélectionnés sont des problèmes de satisfaction de contraintes, manipulant des variables continues, qui sont issus du benchmark Coconut [2] et du benchmark MINLP [3]. Les problèmes *circle*, *mickey* et *exnewton* sont des problèmes classiques d'arithmetique d'intervalles, les problèmes *non-linX* et *pentagon* sont des applications utilisant des systèmes d'équations non linéaires. Le problème *o32* est un problème impliquant des constraintes quadratiques non-convexes.

5.2 Analyse

Le tableau 1 donne les résultats de la méthode de résolution abstraite avec et sans élimination. Pour une précision fixée de $1e^{-3}$ (i.e la taille minimale des éléments retournés), nous nous intéresserons principalement au nombre d'éléments retournés par l'algorithme et au volume couvert correspondant. Les exemples suivants ont été réalisés avec le domaine des intervalles.

Les colonnes 1 et 2 fournissent les informations du problème résolu, à savoir, son nom et, son nombre de variables et de contraintes. Le reste du tableau fournit les informations sur les deux modes de résolution comparés. On a respectivement avec et sans élimination : le temps de résolution en ms (3 et 7, col t), le nombre d'éléments intérieurs (4 et 8, col $\#I$), le nombre d'éléments frontaliers (5 et 9, col $\#E$) et le volume [4] total couvert, (6 et 10, col V). La prise en compte — ou non — d'éléments frontaliers permet d'avoir une résolution respectivement correcte ou complète.

5.2.1 Résultats

On remarque que sur la majorité des problèmes testés, la résolution avec élimination permet de réduire le nombre d'éléments abstraits nécessaires à la couverture de l'espace des solutions. Aussi, notons que ce gain d'éléments ne se fait pas au détriment du volume couvert, celui-ci étant extrêmement proche avec les deux modes de résolution. Aussi, les temps de résolution sont un peu plus longs pour la majorité des exemples ce qui s'explique par le fait que l'on manipule plus d'éléments abstraits lors de la résolution : en effet pour chaque élément, la tehnique d'élimination nécéssite le calcul d'un élément complémentaire. Ces bons résultats viennent confirmer l'intuition que couper un élément par rapport aux contraintes qu'il ne satisfait pas est plus intéressant que de le couper arbitrairement sans tenir compte des contraintes. Enfin, notons ici que l'étape d'élimination permet d'effectuer moins d'étapes de propagation et d'exploration.

6 Conclusion et travaux futurs

Nous avons présenté dans cet article un opérateur de différence ainsi que l'étape associée dans la méthode de résolution. Nous nous somme basés sur la méthode de résolution introduite dans [17] et [18], celle-ci est générique et modulaire, basée entièrement sur les domaines abstraits. Nous y avons intégré un mécanisme d'élimination permettant d'améliorer ses résultats en vertus d'un critère quantitatif. Cette technique qui délaie le choix heuristique de l'exploration permet de mieux tirer profit des contraintes d'un problème pour un temps de calcul très peu supérieur. L'opérateur de différence introduit permet dans certains cas de fortement réduire l'espace de recherche et d'affiner les résultats du solveur. Enfin, soulignons que bien qu'implantée dans

1. `https://github.com/mpelleau/AbSolute`
2. `http://www.mat.univie.ac.at/~neum/glopt/coconut/`
3. `http://www.gamsworld.org/minlp/minlplib/minlpstat.htm`

4. Il s'agit du volume d'un hypercube dont le nombre de dimensions est égal au nombre de variables du problème

problem	$\lvert\mathcal{X}\rvert, \lvert\mathcal{C}\rvert$	sans elimination				avec elimination			
		t	#I	#E	V	t	#I	#E	V
nonlin1	2, 3	**410**	16529	22001	4.551	491	**12474**	18525	4.550
nonlin2	2, 3	**545**	26560	33352	6.088	658	**25510**	30775	6.088
circle	3, 10	128	2136	4903	$5.28 * 10^{-4}$	**125**	**1786**	4906	$5.28 * 10^{-4}$
o32	5, 7	**7922**	1601	103219	$1.4 * 10^{-4}$	10741	848	108889	$1.44 * 10^{-4}$
pentagon	11, 17	**105**	0	11	0.021	200	0	**8**	0.02
booth	2, 2	**342**	8419	10753	1.044	421	8419	10753	1.044
mickey	2, 5	**129**	742	720	2.108	155	**632**	674	2.107
exnewton	2, 3	**158**	7170	7245	0.478	190	**6830**	6834	0.478

TABLE 1 – Comparaison de la méthode de résolution avec et sans élimination

un solveur spécifique à l'utilisation de domaines abstraits, cette technique peut parfaitement être intégrée à un solveur plus classique.

Nous envisageons de déterminer de façon heuristique quand effectuer une étape d'élimination. En effet, lorsque les éléments sont petits, l'étape d'élimination ne conserve que de petites zones solutions et est plus coûteuse qu'elle n'apporte à la résolution. De plus, l'élimination étant plus efficace pour certaines contraintes, l'heuristique pourrait ainsi déterminer pour quelles contraintes il est plus intéressant de calculer le complémentaire. Par exemple, certaines contraintes non convexes peuvent introduire des composantes non consistantes dans l'espace de recherche. Nous pouvons alors utiliser la technique d'élimination pour *localiser* ces zones et mieux orienter la recherche.

Il serait intéressant de comparer les performances de cette technique avec d'autres domaines abstraits et d'autres consistances et heuristiques de coupe. Finalement, nous aimerions pouvoir essayer cette technique au sein d'autres solveurs.

Références

[1] Heikel Batnini, Claude Michel, and Michel Rueher. Mind the gaps : A new splitting strategy for consistency techniques. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709 of *Lecture Notes in Computer Science*, pages 77–91. Springer-Verlag, 2005.

[2] Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers, and Jean-François Puget. Revisiting hull and box consistency. In *Proceedings of the 16th International Conference on Logic Programming*, pages 230–244, 1999.

[3] Julien Bertrane, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival. Static analysis and verification of aerospace software by abstract interpretation. In *AIAA Infotech@Aerospace 2010*, Atlanta, Georgia, 20–22 April 2010. American Institute of Aeronautics and Astronautics.

[4] Christian Bessière and Jean-Charles Régin. Mac and combined heuristics : Two reasons to forsake fc (and cbj ?) on hard problems. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, volume 1118 of *Lecture Notes in Computer Science*. Springer, 1996.

[5] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, (ECAI'2004)*, pages 146–150. IOS Press, 2004.

[6] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4) :251–256, 1979.

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

[8] Hélène Collavizza, François Delobel, and Michel Rueher. Extending consistent domains of numeric csp. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 406–413, 1999.

[9] Patrick Cousot and Radhia 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, 1977.

[10] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 84–96, 1978.

[11] Jérôme Feret. Static analysis of digital filters. In Springer, editor, *European Symposium on Pro-*

gramming (ESOP'04), volume 2986, pages 33–48, 2004.

[12] Eldon Hansen. *Global optimization using interval analysis*. Marcel Dekker, 1992.

[13] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. In *Proceedings of the 6th International Joint Conference on Artificial intelligence (IJCAI'79)*, pages 356–364. Morgan Kaufmann Publishers Inc., 1979.

[14] Bertrand Jeannet and Antoine Miné. Apron : A library of numerical abstract domains for static analysis. In *Proceedings of the 21th International Conference Computer Aided Verification (CAV 2009)*, 2009.

[15] Antoine Miné. *Domaines numériques abstraits faiblement relationnels*. PhD thesis, École Normale Supérieure, December 2004.

[16] Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1) :31–100, 2006.

[17] Marie Pelleau, Antoine Miné, Charlotte Truchet, and Frédéric Benhamou. A constraint solver based on abstract domains. In *Proceedings of the 14th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2013)*, 2013.

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

[19] Dietmar Ratz. Box-splitting strategies for the interval gauss-seidel step in a global optimization method. *Computing*, 53 :337–354, 1994.

A Procédures annexes de la méthode de résolution

A.1 Procédure d'initialisation

Algorithme 3 Procédure d'initialisation

function INIT($\mathcal{D}$) ▷ $\mathcal{D}$: domaines des variables
  $e' \leftarrow \bot$
  for $i \in \mathcal{D}$ **do**
    $e' \leftarrow e' \cup \alpha(i)$
  end for
  return e'
end function

Cette procédure permet l'initialisation de l'algorithme en sur-approximant les domaines des variables. Il s'agit de l'union des abstractions des valeurs possibles des variables.

A.2 Procédure de filtrage

Algorithme 4 Procédure de filtrage

function FILTRE($e, \mathcal{C}$) ▷ e : un élément abstrait $\mathcal{C}$: contraintes
  $e' \leftarrow e$
  for $c_i \in \mathcal{C}$ **do**
    $e' \leftarrow cons_A(e', c_i)$
  end for
  return e'
end function

Cette procédure permet le filtrage d'un élément par rappot à un ensemble de contraintes. On filtre successivement chaque contrainte de l'élément à l'aide de l'opérateur de consistance. Différentes consistances sont possibles. AbSolute implémente l'algorithme `Bottom-Up, Top-down` de [15, §2.4.4] développé indépendamment en PPC sous le nom de `HC4-revise` de [2].

A.3 Test de satisfaction

Algorithme 5 Test de satisfaction de contraintes

function SATISFAIT($e, \mathcal{C}$) ▷ e : un élément abstrait $\mathcal{C}$: contraintes
  for $c_i \in \mathcal{C}$ **do**
    if $cons_A(e, \neg c_i) \neq \bot$ **then**
      return false
    end if
  end for
  return true
end function

Cette procédure prend un élément abstrait et une liste de contraintes et retourne vrai si et seulement si l'élément abstrait satisfait toutes les contraintes. On considère qu'un élément satisfait une contrainte si il possède une intersection nulle avec la négation des contraintes. On pourrait être ici plus précis/efficace en utilisant un test de satisfaction spécifique par domaine abstrait, mais cette méthode, plus générique facilite l'implémentation de nouveaux domaines.

A.4 Complémentaire

Algorithme 6 Complémentaire

function COMPLEMENTAIRE($e, \mathcal{C}$) ▷ e : un élément abstrait $\mathcal{C}$: contraintes
  $e' \leftarrow \emptyset$
  for $c_i \in \mathcal{C}$ **do**
    $e' \leftarrow e' \cup cons_A(e, \neg c_i)$
  end for
  return e'
end function

Cette procédure prend un élément abstrait et un ensemble de contrainte $\mathcal{C}$. Elle retourne un élément e' inclut dans e correspondant à un espace complémentaire de celui definit par les contraintes de $\mathcal{C}$.

ClosedPattern : Une contrainte globale pour l'extraction de motifs fréquents fermés *

M. Maamar[1,2] C. Bessiere[1] P. Boizumault[3] N. Lazaar[1] Y. Lebbah[2] V. Lemière[3] S. Loudni[3]

[1] Laboratoire LIRMM, Université de Montpellier, France

[2] Laboratoire LITIO, Université d'Oran, Algérie

[3] Laboratoire Greyc, Université de Caen, France

nom@lirmm.fr, nom.prénom@univ-oran.dz, prénom.nom@unicaen.fr

Résumé

L'extraction de motifs fréquents fermés est un des défis majeurs en fouille de données. Les travaux entrepris récemment en extraction de motifs ont mis en avant l'intérêt d'utiliser les contraintes pour une fouille déclarative. Ces approches se sont montrées très attractives par leurs flexibilité, mais l'utilisation d'un nombre important de contraintes réifiées et de variables auxiliaires posent un sérieux problème quant au traitement des bases de grandes tailles. Dans ce papier, nous présentons une contrainte globale nommée ClosedPattern , qui capture la sémantique particulière des motifs fermés pour résoudre efficacement ce problème, sans faire appel aux contraintes réifiées. Nous proposons un algorithme de filtrage pour la contrainte ClosedPattern, qui maintient la consistance de domaine DC en un temps et espace polynomial.

1 Introduction

La contrainte globale ClosedPattern a pour objectif de palier à deux problèmes majeurs dans la fouille de motifs, à savoir : (i) La rigidité des algorithme classiques [4, 5] pour la prise en compte de nouvelles contraintes. (ii) La lourdeur du modèle réifié proposé dans [1, 2]. En effet, la contrainte globale ClosedPattern permet d'extraire les motifs fréquents fermés d'une base de transactions donnée, sans faire appel aux contraintes réifiées. L'espace de recherche exploré par la contrainte ClosedPattern, est défini uniquement sur les variables de décision représentant les items, contrairement à l'approche déclarative [1], où le modèle nécessite une autre dimension liée aux variables de transactions. La contrainte ClosedPattern repose sur un algorithme de filtrage efficace assurant la consistance de domaine sur chaque nœud de l'arbre de recherche, avec un temps et un espace polynomiale. L'algorithme de filtrage maintient trois règles de filtrage qui permettent d'élaguer toutes valeurs inconsistantes qui ne mènent pas vers un motif fréquent ou fermés. La résolution établit par la contrainte ClosedPattern est sans retour arrière.

*Ce papier est un résumé de l'article publié à CP'16 : "A Global Constraint for Closed Frequent Patterns Mining" [3]

2 ClosedPattern : Encodage et filtrage

Étant donné n items, soit P le motif fréquent fermé recherché, encodé à l'aide des variables booléennes $P_1, ..., P_n$ représentant les items du motif. La contrainte globale ClosedPattern assure à la fois, la propriété de fréquence minimale et la propriété de fermeture. Soit $\mathcal{D}$ la base de transactions et θ le seuil de fréquence minimale. La contrainte $\text{ClosedPattern}_{\mathcal{D},\theta}(P)$ est satisfaite, si et seulement si, $freq_{\mathcal{D}}(P) \geq \theta \wedge P$ forme un motif fermé.

Afin de satisfaire la contrainte ClosedPattern, nous avons proposé trois règles de filtrage pour caractériser l'inconsistance des valeurs 0/1 pour chaque item.

Règle 1 : : Cette règle prend son origine dans la notion de *merging item* proposée dans [6]. Lorsqu'un item i est présent dans toutes les transactions qui couvrent l'instanciation partielle courante, alors, ce genre d'item doit forcement être présent pour former un motif fermé : $0 \notin D(P_i)$.

Règle 2 : : Cette règle est une règle de base, dérivée de la propriété d'anti-monotonie de la fréquence. Si l'ajout d'un item i à l'instanciation partielle courante

forme un motif non fréquent, alors, cet item doit être absent : $1 \notin D(P_i)$.

Règle 3 : Cette règle est originale, elle tire son raisonnement des items absents du motif courant. Si la couverture d'un item i est un sous-ensemble de celle d'un item k. Ainsi, si l'item k est absent $P_k = 0$, alors, l'item i doit également être absent : $1 \notin D(P_i)$.

3 Complexité

Étant donnée une base de transactions $\mathcal{D}$ avec n items et m transactions, et un support minimal θ. L'algorithme de filtrage de la contrainte ClosedPattern maintient la consistance de domaine, ou prouve l'inconsistance dans un temps $O(n^2 \times m)$ avec une complexité en espace en $O(n \times m)$. L'algorithme de filtrage maintient la DC sur des variables booléennes à chaque nœud, ce qui implique que l'arbre de recherche exploré est un arbre binaire entier. Ainsi, la taille de l' arbre est (2 × nombre de nœuds) - 1. Par conséquent, la complexité totale pour extraire l'ensemble des motifs fréquents fermés, noté $\mathcal{C}$, est en $O(\mathcal{C} \times n^2 \times m)$.

4 Expérimentations

Les expérimentations ont été menées sur une série de bases de transactions issues du dépôt fimi [1]. Les résultats ont montré une domination nette de ClosedPattern par rapport au modèle réifié en terme de nombre de nœuds explorés, de nombre de propagations et de mémoire consommée. Par ailleurs, sur les bases de grandes tailles, le modèle réifié n'est pas capable de charger ces bases en mémoire. En terme de temps de calcul, ClosedPattern domine le modèle réifié, mais reste moins performant que l'algorithme spécialisé lcm.

Dans une seconde étude expérimentale, nous avons étudié une voie prometteuse dans la découverte de motifs ; qui est de poser des contraintes sur un ensemble de k motifs (k-patterns sets). Dans ce contexte, l'intérêt d'un motif est évalué par rapport à un ensemble de motifs. Nous proposons de modéliser et résoudre une instance particulière, où l'objectif est d'extraire les k motifs fermés $\{P_1, ..., P_k\}$ tels que :

(i) $\forall i \in [1, k]$: ClosedPattern(P^i).
(ii) $\forall i, j \in [1, k] : P^i \cap P^j = \emptyset$.
(iii) $\forall i \in [1, k] : lb < |P^i| < ub$.

Cette étude (Fig.1) montre que ClosedPattern a un comportement linéaire en variant k, alors que cp4im suit une échelle exponentielle et va au-delà du timeout sur les deux bases choisies. L'utilisation de lcm avec un post-traitement sur les k combinaisons possibles des motifs fermés est très couteuse et devient rapidement impraticable.

1. http ://fimi.ua.ac.be/data/

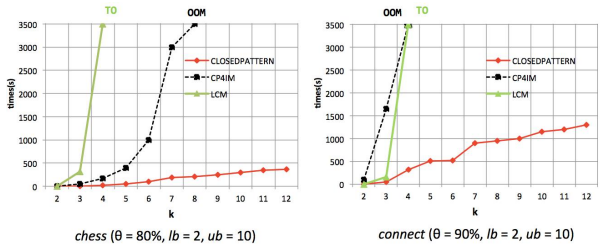


FIGURE 1 – k motifs avec ClosedPattern, cp4im, lcm

5 Conclusion

Nous avons proposé la contrainte ClosedPattern pour l'extraction de motifs fréquents fermés. Afin de propager efficacement la contrainte, nous avons défini trois règles de filtrage qui assurent la DC. Nous avons conçu un algorithme de filtrage, qui établit la DC avec une complexité cubique en temps et quadratique en espace. Nous avons vu, que ClosedPattern offre un apport pratique sur les bases de grande taille, ce qui est un enjeu majeur pour la communauté de fouille.

Références

[1] L. De Raedt, T. Guns, and S. Nijssen. Constraint programming for itemset mining. In *Proceedings of the 14th ACM SIGKDD*, pages 204–212. ACM, 2008.

[2] T Guns, S Nijssen, and L De Raedt. Itemset mining : A constraint programming perspective. *Artificial Intelligence*, 175(12) :1951–1983, 2011.

[3] N. Lazaar, Y. Lebbah, S. Loudni, M. Maamar, V. Lemière, C. Bessiere, and P. Boizumault. *A Global Constraint for Closed Frequent Pattern Mining*, pages 333–349. CP'16, 2016.

[4] J. Pei, J. Han, and R. Mao. CLOSET : an efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop*, pages 21–30, 2000.

[5] T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *DS 2004, Italy, 2004, Proceedings*, pages 16–31, 2004.

[6] J. Wang, J. Han, and J. Pei. CLOSET+ : searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the 9th ACM SIGKDD*, pages 236–245, 2003.

Étude de la modélisation en programmation par contraintes pour résoudre le problème de localisation/routage

Laure Brisoux-Devendeville **Corinne Lucet**

Laboratoire de Modélisation, Information et Systèmes EA 4290
Université de Picardie Jules Verne,
33 rue Saint Leu - 80039 Amiens Cedex 1 - France
{laure.devendeville,corinne.lucet}@u-picardie.fr

Résumé

Le problème de localisation routage (Location Routing Problem ou LRP) est un problème de logistique qui peut être vu comme la combinaison de deux problèmes de décision difficiles, à savoir le problème de tournées de véhicules (Vehicle Routing Problem ou VRP) et le problème de placement de ressources (Facility Location Problem ou FLP). Le LRP est un problème d'optimisation combinatoire NP-difficile largement étudié dont le but est d'optimiser un coût total combinant le coût des dépôts ouverts et le coût des routes empruntées. Chaque dépôt est associé à un unique véhicule, chargé de desservir un sous ensemble de clients, en respectant des contraintes de capacités. Tous les clients doivent être servis. Dans cet article, nous proposons une modélisation du LPR par Programmation par Contraintes. Le but est de comparer cette approche, à la résolution du LRP par la Programmation Linéaire Mixte en Nombres Entiers. Ce travail est une première étape prospective sur la pertinence de l'emploi de CSP pour résoudre le LRP, pour lequel l'élaboration de bornes est difficile par nature. L'évaluation expérimentale sera mise en œuvre sur des benchmarks de la littérature.

Abstract

Location routing problem (LRP) is a problem of logistic that can be seen as the combination of two difficult decision problems : the vehicle routing problem (VRP) and the facility location problem (FLP). LRP is widely studied NP-hard problem whose goal is to optimize a total cost combining the cost of open deposits and the cost of selected roads. Each deposit has a single associated vehicle that serves a subset of customers while respecting capacity constraints. All customers must be served. In this paper we propose to model the LPR by Constraint Programming and compare this approach with a Mixed Linear Programming model. This work is a first prospective step on the relevance of using CSP to solve the LRP. Experimental evaluations will be done on benchmarks from the LRP literature.

1 Location-Routing Problem

L'évolution des modes de consommation, le développement du commerce électronique, mais aussi les changements de types de productions industrielles (séries plus courtes) ont favorisé dernièrement le développement du transport de marchandises. La gestion de cette activité soulève le problème de la construction d'un système de distribution efficace. Ceci comprend le choix des dépôts où les marchandises sont stockées ainsi que le calcul des tournées de véhicules nécessaires à la distribution de ces marchandises. Ces deux points sont étroitement liés et si une solution optimale doit être trouvée, ils ne peuvent être résolus séparément. Ce problème est un problème NP-difficile que l'on trouve sous le nom de « problème de localisation-routage »(location routing problem - LRP) dans la littérature. Il est généralement traité soit, de manière approchée par des heuristiques et/ou des métaheuristiques comme les algorithmes évolutifs pour les instances de grandes tailles [1, 6, 7], soit de manière exacte sous forme de Programmation Linéaire Mixte en Nombres Entiers (PLM) ou encore par des techniques de branch-and-bound pour les instances plus petites [8, 3, 4].

Dans les multiples formes que peut prendre le LRP, différents paramètres sont considérés comme les capacités et le nombre de véhicules disponibles (la flotte), les capacités des dépôts, la disponibilité des clients,

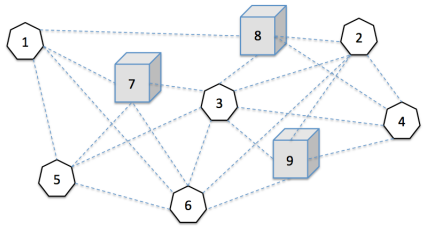


FIGURE 1 – Exemple de problème avec 3 dépôts et 6 clients

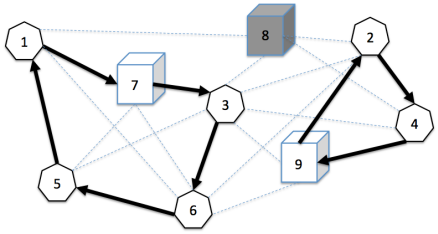


FIGURE 2 – Exemple de solution

certaines contraintes de synchronisation, etc. Dans cet article, nous supposons que nous avons un véhicule disponible sans limite de capacité pour chaque dépôt qui lui possède une contrainte de capacité. Nous disposons donc d'un ensemble $I = \{1, \ldots, n\}$ de clients toujours disponibles à servir et d'un ensemble $K = \{1, \ldots, m\}$ de dépôts. À chaque client $i \in I$ est associée une demande Q_i. Chaque dépôt $k \in K$ possède un coût d'ouverture O_k et une capacité maximale C_k. Les routes considérées relient les clients entre eux et les dépôts aux clients. On note $A = (I \times I) \cup (I \times K) \cup (K \times I)$ l'ensemble de ces routes. Chaque route $(i, j) \in A$ possède un coût d'emprunt $D_{i,j}$. La capacité maximale d'un dépôt peut être dépassée avec une pénalité unitaire fixée par la constante α. La figure 1 présente un exemple, comprenant 6 clients et 3 dépôts ainsi que les routes possibles. Une solution possible à cet exemple est présentée figure 2 dans laquelle le dépôt 8 est fermé.

Nous présentons dans cet article un travail préliminaire sur une modélisation du LRP, sous forme de CSP afin de le comparer au modèle mathématique PLM. Nous utiliserons le solveur Choco[10] pour résoudre un ensemble d'instances de la littérature. Nous comparerons ces résultats à ceux que nous avons obtenus par une modélisation du LRP en PLM dont la résolution a été confiée au solveur CPLEX[5].

2 Modèle CSP

Notre modèle de CSP est directement inspiré du modèle de De Backer *et al.* [2] pour des problèmes de tournées de véhicules. Pour le LRP, nous devons intégrer la multiplicité des dépôts et donc des tournées. À chaque dépôt ouvert correspond une tournée. Pour la construction de ces tournées, nous considérerons que chaque dépôt est dédoublé en deux sites : $start_k$ qui définit le début de la tournée associée au dépôt k et end_k la fin. *start* et *end* sont respectivement indexés par $\{n+1 \ldots n+m\}$ et $\{n+m+1 \ldots n+2m\}$. Pour plus de lisibilité, les ensembles d'index sont définis comme suit :

- $N = \{1 \ldots n\}$ (les clients) ;
- $M = \{1 \ldots m\}$ (les dépôts) ;
- $S = \{n+1 \ldots n+m\}$ (les départs de dépôt) ;
- $E = \{n+m+1 \ldots n+2m\}$ (les arrivées de dépôt) ;
- $T = N \cup S \cup E$ (ensemble de tous les sites).

2.1 Variables

Nous utiliserons dans ce modèle trois types de variables : pour la construction des tournées, pour l'affectation des dépôts et celles afférentes à la fonction objectif.

Très classiquement, les variables basiques de succession $succ_i$ donnent le successeur direct de i dans la tournée à laquelle il appartient. $succ_i \in N \cup E,\ \forall i \in N \cup S$. Pour énoncer les contraintes de sous-tours qui évitent la formation de sous-cycles à l'intérieur d'une tournée (i.e. cycles sans dépôt), nous introduisons les variables θ_i qui donnent à chaque client et à chaque dépôt (dédoublé), un numéro d'ordre : $\theta_i \in [1..n+m],\ \forall i \in N \cup E$, et $\theta_i = 0,\ \forall i \in S$.

Pour les affectations des clients aux dépôts, la variable $bydep_i \in M$ donne l'index du dépôt dont la tournée associée dessert le client $i \in N$. On trouve donc dans *bydep* l'ensemble des dépôts ouverts.

Les variables associées à la fonction objectif sont les suivantes : q_i est la quantité cumulée des demandes satisfaites dans la tournée après avoir servi le client $i \in T$ avec $q_i = 0,\ \forall i \in S$ et d_i est la distance parcourue dans la tournée pour arriver jusqu'au client i. $d_i \in [1 \ldots D_{MAX}],\ \forall i \in N \cup E$ et $d_i = 0, \forall i \in S$. D_{MAX} est une borne supérieure de la distance maximale parcourue : $D_{MAX} = \sum_{i \in J \cup I} \max\{D_{i,j}; \forall (i,j) \in A\}$. La distance totale de la tournée associée au dépôt k est donc représentée par d_{n+m+k}.

2.2 Fonction objectif

Le coût qui doit être minimisé dans le LRP est la somme des coûts associés aux distances parcourues, plus la somme des coûts associés à l'ouverture des dépôts plus la somme des pénalités de dépassement de capacité des dépôts ouverts. La somme des distances parcourues est également la somme des longueurs de

toutes les tournées. La fonction de coût est formellement exprimée dans l'équation 1.

$$\min \sum_{k \in bydep} d_{n+m+k} + \sum_{k \in bydep} O_k + \alpha \times \sum_{k \in bydep} \max(q_{n+m+k} - C_k, 0) \quad (1)$$

2.3 Les contraintes

Les contraintes de notre modèle reprennent les deux sous-problèmes du LRP, le problème de routage d'une part et le problème d'affectation des clients aux dépôts d'autre part. À cela s'ajoute les contraintes cumulatives de livraison pour les violations de capacité des dépôts et de longueur des tournées, qui toutes deux interviennent dans la fonction objectif.

Cette première famille de contraintes (équations 2 et 3) intègre les contraintes basiques de succession pour la construction des tournées et la conservation du flot, ainsi que les contraintes permettant l'exclusion des sous-tours.

$$\text{AllDifferent}(\{succ_i | i \in N \cup S\}) \quad (2)$$

$$\begin{aligned} &\text{AllDifferentExcept0}(\{\theta_i | i \in N \cup E\}) \\ &\theta_i = 0, \ \forall i \in S \\ &\theta_{succ_i} > \theta_i, \ \forall i \in N \cup S \end{aligned} \quad (3)$$

Les contraintes d'affectation des clients aux dépôts (équation 4) permettent de s'assurer que toutes les tournées partent et arrivent à un même dépôt.

$$\begin{aligned} bydep_{n+k} &= k, \ \forall k \in M \\ bydep_{n+m+k} &= k, \ \forall k \in M \\ bydep_{succ_i} &= bydep_i, \ \forall i \in N \cup S \end{aligned} \quad (4)$$

Les contraintes cumulatives de livraison (équation 5) permettent de calculer, pour chaque tournée, la quantité totale de demandes servies, qui, dans notre problème peut dépasser la capacité maximale du dépôt concerné.

$$\begin{aligned} q_i &= 0, \ \forall i \in S \\ q_{succ_i} &= q_i + Q_{succ_i}, \ \forall i \in N \cup S \end{aligned} \quad (5)$$

Les contraintes cumulatives des distances parcourues (équation 6) permettent d'évaluer le coût du routage.

$$\begin{aligned} d_i &= 0, \ \forall i \in S \\ d_{succ_i} &= d_i + D_{i,succ_i}, \ \forall i \in N \cup S \end{aligned} \quad (6)$$

L'objectif de cette étude est l'utilisation de ce paradigme de programmation qu'est la programmation par contrainte, pour résoudre le LRP. Nous le comparerons au modèle de PLM de la section suivante.

3 Modèle Programme Linéaire Mixte

Le programme linéaire exposé dans ce paragraphe est celui que nous utiliserons pour comparer les deux approches de modélisation et de résolution. Il est issu des travaux de synthèse proposés dans [9]. Les variables sont de type entier et booléen. Nous utilisons pour celui-ci les mêmes données de base, énoncées dans la section 1.

3.1 Variables et Notations

Les variables

- $x_e = 1$ Si la route $e \in A$ est utilisée
- $u_k = 1$ Si le dépôt $k \in K$ est ouvert
- $y_{ik} = 1$ Si le client $i \in I$ est servi par le dépôt $k \in K$
- $\theta_i \in [1, |I|]$: ordre du client $i \in I$ dans sa tournée. Ces variables, comme pour le CSP, permettent de ne pas engendrer de sous-tours.

Les notations

- Soit $V' \subseteq V$, $\delta^+(V')$ est l'ensemble des arcs $e = (i, j)$ avec $i \in V'$ et $j \notin V'$
- On écrira $\delta^+(i)$ plutôt que $\delta^+(\{i\})$ si $V' = \{i\}$.
- $\forall e \in A, \ e = (i, j), \ \pi^+(e) = j$

3.2 Fonction objectif

La fonction objectif est de même composition que celle énoncée dans le CSP. Elle est le cumul du coût lié au déplacements, du coût d'ouverture des dépôts et de la somme des pénalités pour dépassement de capacité des dépôts. L'équation 7 représente ce coût global à minimiser.

$$\min \sum_{e \in A} x_e.d_e + \sum_{k \in K} u_k.O_k + \alpha \times \sum_{k \in K} \max(\sum_{i \in I} y_{ik}.q_i - C_k, 0) \quad (7)$$

3.3 Les contraintes

Comme pour le CSP, les contraintes peuvent être groupées par famille relativement au sous-problème de routage ou d'affectation auquel elles se réfèrent. Les équations de 8 à 12 modélisent les contraintes d'affectations des clients aux dépôts. Elles assurent respectivement qu'un client n'est servi que par un seul dépôt, qu'un dépôt ouvert sert au moins un client, qu'un dépôt fermé ne sert aucun client. Pour les deux dernières elles assurent la consistance de l'affectation par rapport aux tournées, à savoir que tous les clients d'une

même tournée sont affectés à un même dépôt.

$$\forall i \in I,\ \sum_{k \in K} y_{ik} = 1 \tag{8}$$

$$\forall k \in K,\ \sum_{i \in I} y_{ik} \geq u_k \tag{9}$$

$$\forall k \in K,\ \forall i \in I,\ y_{ik} \leq u_k \tag{10}$$

$$\forall (i,j) \in I \times I,\ \forall k \in K,\ y_{ik} - y_{jk} \leq 1 - x_{ij} \tag{11}$$
$$\forall (i,j) \in I \times I,\ \forall k \in K,\ y_{jk} - y_{ik} \leq 1 - x_{ij}$$

$$\forall i \in I,\ \forall k \in K,\ x_{ik} + x_{ki} \leq 2 \times y_{ik} \tag{12}$$

Les équations de 13 à 16 modélisent les contraintes de routage. Elles assurent respectivement qu'une seule route arrive et part d'un site client, que les tournées ne sont construites qu'à partir de dépôts ouverts et enfin qu'il n'y ait pas de sous-tours dans ces tournées.

$$\forall i \in I,\ \sum_{j \in V \setminus \{i\}} x_{ij} = 1 \tag{13}$$

$$\forall j \in I,\ \sum_{i \in V \setminus \{j\}} x_{ij} = 1 \tag{14}$$

$$\forall k \in K,\ \sum_{i \in I} (x_{ik} + x_{ki}) = 2.u_k \tag{15}$$

$$\forall i \in I,\ \forall e \in \delta^+(i) \cap A',\ \theta_i + x_e \leq \theta_{\pi^+(e)} + |I|.(1 - x_e) \tag{16}$$

4 Expérimentations et perspectives

Notre objectif est l'implémentation du modèle CSP avec le solveur Choco[10]. Ce modèle sera dans un premier temps testé sur les instances proposées par Albareda *et al* [1]. Ce benchmark pour le LRP est composé de 450 instances avec un nombre de clients variant de 10 à 30, et un nombre de dépôts de 5 à 10. La pénalité unitaire de dépassement de capacité est fixée à $\alpha = 1000$ ([1]). Il sera nécessaire de définir une stratégie de recherche pertinente. Nous comparerons les résultats obtenus avec Choco aux solutions (et bornes) atteintes par CPLEX[5] puisque c'est, sur ces instances, que nous avons le plus de points de comparaisons de part leurs tailles. Nous pourrons ensuite étendre nos expérimentations aux instances proposées par Tuzun et Burke [11] avec de, 100 à 200 clients, et de 10 à 20 dépôts et les instances de Prodhon et Prins [9] avec de, 20 à 200 clients, et de 5 à 10 dépôts.

Références

[1] M. Albareda-Sambola, J.A. Diaz, and E. Fernàndez. A compact model and tight bounds for combined location-routing problem. *Computers and operations Research*, 32(3) :407–428, 2005.

[2] Bruno de Backer, Vincent Furnon, Paul Shaw, Philip Kilby, and Patrick Prosser. Solving vehicle routing problems using constraint programming and metaheuristics. *Journal of Heuristics*, 6(4) :501–523, 2000.

[3] J.-M Belenguer, E. Benavent, C. Prins, C. Prodhon, and R. Wolfler-Calvo. A branch-and-cut method for the capacitated location-routing problem. *Computer and Operation Research*, 38(6) :931–941, 2011.

[4] C. Contardo, J.-F. Cordeau, and B. Gendron. A branch-and-cut-and-price algorithm for capacited location-routing problem. *INFORMS Journal on Computing*, 2013.

[5] IBM ILOG CPLEX Optimizer V12.5. **http ://www-01.ibm.com /software /integration /optimization/cplex-optimizer/**, Last 2014.

[6] H. Derbel, B. Jarboui, S. Hanafi, and H. Chabchoub. An iterated local search for solving a location-routing problem. *Electronic Notes in Discrete Mathematics*, 36 :875–882, 2010.

[7] H. Derbel, B. Jarboui, S. Hanafi, and H. Chabchoub. Genetic algorithm with iterated local search for solving a location-routing problem. *Expert systems with Applications*, 39(3) :2865–2871, 2012.

[8] G. Laporte, Y. Nobert, and D. Arpin. An exact algorithm for solving a capacitated location-routing problem. *Annals of Operations Research*, 6(9) :291–310, 1986.

[9] C. Prodhon and C. Prins. A survey of recent research on location-routing problems. *European Journal of Operational Research*, 238 :1–17, 2014.

[10] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016.

[11] D. Tuzun and L. Burke. A two-phase tabu search approach to the location routing problem. *European Journal of Operational Research*, 116 :87–99, 1999.

Une Approche Basée Sur SAT Pour l'Énumération Des Règles d'Association Non Redondantes

Abdelhamid Boudane *__Said Jabbour Lakhdar Sais Yakoub Salhi__

CRIL-CNRS, Université d'Artois, F-62307 Lens Cedex, France

{boudane,jabbour,sais,salhi}@cril.fr

Résumé

L'extraction des règles d'association à partir de bases de données transactionnelles est une tâche bien étudiée en fouille de données. Plusieurs algorithmes efficaces ont été proposés. Cependant, le nombre de règles extraites peut être énorme. Plusieurs travaux ont attaqué l'extraction d'un petit ensemble de règles d'association jugées les plus pertinentes. Dans cet article, nous adressons le problème d'énumération des règles d'association minimales non redondantes largement considéré comme étant l'une des variantes les plus intéressantes. Nous présentons d'abord son encodage en formule propositionnelle dont les modèles correspondent aux règles minimales non redondantes. Ensuite, nous montrons que l'ensemble des générateurs minimaux, utilisé dans l'extraction des règles non redondantes, peut également être encodé dans ce cadre. Des expérimentations sur de nombreux jeux de données montrent que notre approche permet d'obtenir de meilleures performances par rapport aux techniques spécialisées.

Abstract

Discovering association rules from transaction databases is a well studied data mining task. Many effective techniques have been proposed over the years. However, due to the huge size of the output, many works have tackled the problem of mining a smaller and relevant set of rules. In this paper, we address the problem of enumerating the minimal non-redundant association rules, widely considered as one of the most relevant variant. We first provide its encoding as a propositional formula whose models correspond to the minimal non redundant rules. Then we show that the set of minimal generators used for extracting non-redundant rules can also be encoded in this framework. Experiments on many datasets show that our approach achieves better performance with respect to the state-of-the-art specialized techniques.

*Papier doctorant : Abdelhamid Boudane est auteur principal.

1 Introduction

La fouille des règles d'association à partir des bases de données transactionnelles est un problème qui a reçu beaucoup d'attention depuis son introduction par Rakesh Agrawal et al. dans [1]. Après sa première application sur l'analyse des paniers de consommation, plusieurs autres domaines d'application ont été identifiés comme la bioinformatique, le diagnostique médical, la détection d'intrusion, la fouille du web, l'analyse des documents et l'analyse des données scientifiques. Ce large spectre d'applications a permis à ce type d'analyse d'être appliquée sur une variété de jeux de données à savoir les données séquentielles, spatiales et les graphes. L'extraction des règles d'association est une tâche très intéressante car elle est maintenant considérés comme un élément constitutif de plusieurs autres problèmes d'apprentissage tels que la classification, la régression et le clustering.

La plupart des approches ont mentionné que la tâche classique de fouille de règles d'association produit un très grand nombre de règles [2, 5, 6, 14, 18]. L'énorme taille de cet ensemble de règles extraites n'aide pas l'utilisateur à récupérer facilement des informations pertinentes. Cette observation a conduit à plusieurs définitions de la redondance qui ont été utilisées afin de limiter le nombre de règles d'association. De ce fait, de nombreuses contributions se sont concentrées sur l'élimination des règles redondantes tout en maintenant l'ensemble des règles pertinentes appelées règles d'association non-redondantes (minimales). Plusieurs types de règles non redondantes ont été introduits tels que la base générique [2], La base informative [2], la base informative et générique [6], Conditions minimales et Conséquences maximales (MMR) [14] et l'ensemble des règles représentatives [13] qui couvre

toutes les règles d'association. Pour éliminer les règles redondantes, la plupart des approches partagent les deux étapes suivantes : (1) trouver l'ensemble des générateurs minimaux et les itemsets fermés. (2) générer les règles confiantes en considérant les deux ensembles déjà extraits dans la première étape.

Récemment, des approches déclaratives ont été proposées pour aborder plusieurs tâches de fouille de données à travers la programmation par contraintes (CP) et la satisfiabilité propositionnelle (SAT) [7, 8, 11, 12, 15]. Dans [3], les auteurs ont proposé un nouveau cadre pour fouiller les règles d'association en une seule étape en utilisant la satisfiabilité propositionnelle conduisant à une approche concurrentielle par rapport aux techniques spécialisées. Encouragés par ces résultats, Nous proposons dans cet article d'étendre ce cadre pour extraire les règles minimales non redondantes. La redondance est éliminée élégamment en utilisant de nouvelles contraintes combinées à d'autres listées dans [3]. Nous montrons que deux types de règles non redondantes peuvent être abordées. De plus, une restriction de notre encodage peut être utilisée pour extraire les générateurs minimaux.

2 Préliminaires

2.1 Logique Propositionnelle et Problème SAT

Nous définissons maintenant la syntaxe et la sémantique de la logique propositionnelle. Soit Prop un ensemble dénombrable de variables propositionnelles. Nous utilisons les lettres p, q, r, etc pour noter les éléments de Prop. L'ensemble de *formules propositionnelles*, noté Form, est défini par induction à partir de Prop, la constante $\bot$ désignant le faux, la constante $\top$ désignant le vrai et en utilisant les connecteurs logiques $\neg$, $\wedge$, $\vee$, $\rightarrow$. On utilise $Var(\phi)$ pour noter l'ensemble des variables propositionnelles apparaissant dans une formule ϕ. Le connecteur d'équivalence $\leftrightarrow$ est défini par $\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$.

Une formule ϕ dans une *forme normale conjonctive (CNF)* est une conjonction de clauses. Une *clause* est une disjonction de littéraux et un *littéral* est une variable propositionnelle positive (p) ou négative ($\neg p$). Les deux littéraux p et $\neg p$ sont appelés *complémentaires*.

Une *interprétation* $\mathcal{I}$ d'une formule propositionnelle ϕ est une fonction qui associe à chaque variable $p \in Var(\phi)$ une valeur $\mathcal{I}(p) \in \{0, 1\}$ (0 correspond à $faux$ et 1 à $vrai$). Un *modèle ou un impliquant* d'une formule ϕ est une interprétation $\mathcal{I}$ qui satisfait ϕ c-à-d $\mathcal{I}(\phi) = 1$. Le problème *SAT* consiste à décider si une formule CNF donnée admet un modèle ou non.

2.2 Règles d'Association

Soit Ω un ensemble fini non vide de symboles, appelés *items*. A partir de maintenant, nous supposons que cet ensemble est fixé. Nous utilisons les lettres a, b, c, etc pour noter les éléments de Ω. Un *itemset* I sur Ω est défini comme étant un sous ensemble de Ω, c-à-d $I \subseteq \Omega$. Nous utilisons 2^Ω pour désigner l'ensemble des itemsets sur Ω et nous utilisons les lettres majuscules I, J, K, etc pour noter les éléments de 2^Ω.

Une *transaction* est une paire ordonnée (i, I) où i est un nombre naturel appelé *identifiant* et I est un itemset, c-à-d $(i, I) \in \mathbb{N} \times 2^\Omega$. Une *base de données transactionnelles* $\mathcal{D}$ est un ensemble fini non vide de transactions ($\mathcal{D} \subseteq \mathbb{N} \times 2^\Omega$) où chaque identifiant fait référence à un itemset unique.

Étant donnée une base de données transactionnelles $\mathcal{D}$ et un itemset I, la *couverture* de I dans $\mathcal{D}$, notée $\mathcal{C}(I, \mathcal{D})$, est définie comme suit : $\{i \in \mathbb{N} \mid (i, J) \in \mathcal{D}$ *et* $I \subseteq J\}$. Le *support* de I dans $\mathcal{D}$, noté $Supp(I, \mathcal{D})$, correspond à la cardinalité de $\mathcal{C}(I, \mathcal{D})$ c-à-d $Supp(I, \mathcal{D}) = |\mathcal{C}(I, \mathcal{D})|$. Un itemset $I \subseteq \Omega$ tel que $Supp(I, \mathcal{D}) \geqslant 1$ est un itemset *fermé* si pour tout itemset J avec $I \subset J$, $Supp(J, \mathcal{D}) < Supp(I, \mathcal{D})$.

Exemple 1 *Soit la base de données transactionnelles $\mathcal{D}$ représentée dans la Table 1. On a $\mathcal{C}(\{c, d\}, \mathcal{D}) = \{1, 2, 3, 4, 5\}$ et $Supp(\{c, d\}, \mathcal{D}) = 5$ tandis que $Supp(\{f\}, \mathcal{D}) = 3$. L'itemset $\{c, d\}$ est fermé alors que $\{f\}$ ne l'est pas car $Supp(\{f\}, \mathcal{D}) = Supp(\{c, d, f\}, \mathcal{D})$.*

tid	Transactions
1	$c\ d\ e\ f\ g$
2	$c\ d\ e\ f\ g$
3	$a\ b\ c\ d$
4	$a\ b\ c\ d\ \ f$
5	$a\ b\ c\ d$
6	$c\ \ e$

TABLE 1 – Base de Données Transactionnelles $\mathcal{D}$

Nom	Règles d'association	Support	Confiance
r_1	$\{a\} \rightarrow \{b\}$	3/6	1
r_2	$\{a\} \rightarrow \{b, c, d\}$	3/6	1
r_3	$\{c\} \rightarrow \{d\}$	5/6	5/6
r_4	$\{c, d\} \rightarrow \{e, f, g\}$	2/6	2/5

TABLE 2 – Quelques Règles d'Association

Dans ce travail on s'intéresse au problème de fouille de règle d'association (MAR). Une *règle d'association* est un pattern de la forme $X \rightarrow Y$ où X (appelé antécédent) et Y (appelé conséquence) sont deux itemsets

disjoints. Dans MAR, le prédicat qui mesure l'importance d'une règle est défini en utilisant les notions de support et de confiance. Le *support* d'une règle d'association $X \rightarrow Y$ dans une base de données transactionnelles $\mathcal{D}$ est $Supp(X \rightarrow Y, \mathcal{D}) = \frac{Supp(X \cup Y, \mathcal{D})}{|\mathcal{D}|}$. Il permet de déterminer combien une règle est applicable dans une base $\mathcal{D}$, c-à-d la fréquence d'occurrence de la règle. La *confiance* de $X \rightarrow Y$ dans $\mathcal{D}$ est définie comme suit : $\mathcal{C}onf(X \rightarrow Y, \mathcal{D}) = \frac{Supp(X \cup Y, \mathcal{D})}{Supp(X, \mathcal{D})}$. Elle donne une estimation de la probabilité conditionnelle de Y sachant X. A partir de maintenant, on utilise $Supp(X \rightarrow Y)$ et $\mathcal{C}onf(X \rightarrow Y)$ pour noter le support et la confiance de la règle $X \rightarrow Y$.

Une *règle d'association valide* est une règle avec un support et une confiance supérieurs ou égaux au seuil de support minimal (minsupp) et au seuil de confiance minimum (minconf) respectivement. Etant donné une base de données transactionnelles $\mathcal{D}$, un seuil minimum de support minsupp et un seuil minimum de confiance minconf, le problème de fouille de règles d'association consiste à calculer l'ensemble suivant : $\mathcal{MAR}(\mathcal{D}, minsupp, minconf) = \{X \rightarrow Y \mid X, Y \subseteq \Omega,\ Supp(X \rightarrow Y, \mathcal{D}) \geqslant minsupp,\ \mathcal{C}onf(X \rightarrow Y, \mathcal{D}) \geqslant minconf\}$

La Table 2 illustre certaines règles d'association avec leurs supports et confiances correspondants. Par exemple, $Supp(\{a\} \rightarrow \{b\}) = \frac{3}{6}$ et $\mathcal{C}onf(\{a\} \rightarrow \{b\}) = 1$.

3 Fouille de Règles d'Association Basée sur SAT

Dans cette section, nous examinons brièvement l'approche récente proposée dans [3] pour la fouille des règles d'association via la satisfiabilité propositionnelle (SAT). L'idée de base consiste à modéliser cette tâche de fouille en une formule propositionnelle dont les modèles correspondent exactement aux règles d'association recherchées. Dans cet encodage, deux ensembles de variables booléennes sont utilisés pour représenter les items des règles d'association $X \rightarrow Y$ et les transactions. Ensuite, le support et la confiance d'une règle d'association sont capturés par des inéquations linéaires sur les variables booléennes associées aux transactions. Afin de définir l'encodage basé sur SAT, nous fixons, sans perte de généralité, un ensemble Ω de n items, une base de données transactionnelles $\mathcal{D} = \{(1, I_1), \ldots, (m, I_m)\}$ où $\forall i \in \{1, m\}, I_i \subseteq \Omega$, un seuil minimum de support $minsupp$ et un seuil minimum de confiance $minconf$.

Pour capturer les deux parties de chaque règle d'association, nous associons à chaque item a deux variables booléennes notées x_a et y_a. Les variables de la forme x_a (resp. y_a) Sont utilisées pour représenter l'antécédent (resp. conséquent) de chaque règle candidate. Ensuite, pour représenter la couverture de X et $X \cup Y$, chaque identifiant de transaction $i \in \{1, m\}$ est associé à deux variables propositionnelles p_i et q_i. Les variables de la forme p_i (resp. q_i) sont utilisées pour représenter la couverture de X (resp. $X \cup Y$). Plus précisément, étant donnée une interprétation booléenne $\mathcal{B}$, la règle d'association correspondante, notée $r_{\mathcal{I}}$, est $X = \{a \in \Omega \mid \mathcal{I}(x_a) = 1\} \rightarrow Y = \{b \in \Omega \mid \mathcal{I}(y_b) = 1\}$, la couverture de X est $\{i \in \{1, m\} \mid \mathcal{I}(p_i) = 1\}$, et la couverture de $X \cup Y$ est $\{i \in \{1, m\} \mid \mathcal{I}(q_i) = 1\}$. L'encodage SAT du problème d'énumération des règles d'association consiste en un ensemble de contraintes définies comme suit.

$$(\bigvee_{a \in \Omega} x_a) \wedge (\bigvee_{a \in \Omega} y_a) \tag{1}$$

$$\bigwedge_{a \in \Omega} (\neg x_a \vee \neg y_a) \tag{2}$$

$$\bigwedge_{i \in 1..m} \neg p_i \leftrightarrow \bigvee_{a \in \Omega \setminus I_i} x_a \tag{3}$$

$$\bigwedge_{i \in 1..m} \neg q_i \leftrightarrow \neg p_i \vee (\bigvee_{a \in \Omega \setminus I_i} y_a) \tag{4}$$

$$\sum_{i \in 1..m} q_i \geqslant m \times minsupp \tag{5}$$

$$\frac{\sum_{i \in 1..m} q_i}{\sum_{i \in 1..m} p_i} \geqslant minconf \tag{6}$$

Les deux clauses de la formule 1 expriment que X et Y ne sont pas des ensembles vides. La formule (2) permet d'exprimer $X \cap Y = \emptyset$. Elle est simplement définie en imposant que les deux variables x_a et y_a ne peuvent pas être vraies en même temps pour chaque item a. La troisième contrainte est utilisée pour représenter la couverture de l'itemset correspondant à la partie gauche de la règle d'association candidate. Étant donné un itemset X, nous savons que l'identifiant de la transaction i n'appartient pas à $\mathcal{C}(X, \mathcal{D})$ si et seulement s'il existe un item $a \in X$ tel que $a \notin I_i$. Cette propriété est représentée par la contrainte (3) qui exprime que p_i est $faux$ si et seulement si X contient un item qui n'appartient pas à la transaction i. De la même façon, la formule (4) permet de capturer la couverture de $X \cup Y$.

Pour préciser que le support de la règle candidate doit être supérieur ou égal au seuil fixé $minsupp$ (en pourcentage), et que la confiance est supérieure ou égale à $minconf$, nous utilisons les contraintes (5) et (6) exprimées par des inéquations linéaires 0/1.

Pour étendre la tâche de fouille au règles d'association fermées, les contraintes suivantes sont ajoutées pour exprimer que $X \cup Y$ est un itemset fermé [9] :

$$\bigwedge_{a \in \Omega} ((\bigwedge_{i \in 1..m} q_i \rightarrow a \in I_i) \rightarrow x_a \vee y_a) \quad (7)$$

Cette formule signifie que, pour tout item $a \in \Omega$, si nous avons $\mathcal{C}(X \cup Y, \mathcal{D}) = \mathcal{C}(X \cup Y \cup \{a\}, \mathcal{D})$, qui est encodé avec la formule $\bigwedge_{i \in \{1,m\}} q_i \rightarrow a \in I_i$, alors nous obtenons $a \in X \cup Y$, qui est encodé avec $x_a \vee y_a$.

4 Règles d'Association Minimales Non Redondantes

Dans cette section, nous présentons notre encodage du problème d'extraction des règles non-redondantes vers la satisfiabilité propositionnelle. D'abord, Nous nous concentrons sur la représentation intéressante qui correspond au règles d'association minimales non redondantes (MNRs) [14, 2].

Définition 1 *Une règle d'association $r : X \rightarrow Y$ est une règle minimale non-redondante ssi il n'y a pas de règle d'association $r' : X' \rightarrow Y'$ différente de r telle que (i) $Supp(r) = Supp(r')$, (ii) $Conf(r) = Conf(r')$ et (iii) $X' \subseteq X$ et $Y \subseteq Y'$.*

Exemple 2 *Considérons à nouveau les règles d'association données dans la table 2. Dans cet ensemble de règles, $r_2 : \{a\} \rightarrow \{b, c, d\}$ est une règle minimal non redondante tandis que $r_1 : \{a\} \rightarrow \{b\}$ ne l'est pas.*

Dans la proposition 1, nous soulignons que toutes les règles d'association minimales non redondantes sont fermées.

Proposition 1 *Si $r : X \rightarrow Y$ est une règle minimale non redondante dans une base de données transactionnelles $\mathcal{D}$ alors $X \cup Y$ est un itemset fermé dans $\mathcal{D}$.*

Preuve 1 *Supposons que $X \cup Y$ n'est pas un itemset fermé. Alors, il existe un item $a \notin X \cup Y$ tel que $Supp(X \cup Y, \mathcal{D}) = Supp(X \cup Y \cup \{a\}, \mathcal{D})$. Considérons maintenant la règle $r' : X \rightarrow Y \cup \{a\}$. Nous obtenons clairement $Supp(r) = Supp(r')$ et $Conf(r) = Conf(r')$ puisque $Supp(X \cup Y, \mathcal{D}) = Supp(X \cup Y \cup \{a\}, \mathcal{D})$. Ainsi, r n'est pas une règle minimale non redondante et nous obtenons une contradiction.*

Autrement dit, les règles d'association minimales non redondantes sont des règles fermées dans lesquelles les antécédents sont minimales par rapport à l'inclusion ensembliste. En utilisant cette propriété, les auteurs de [2] ont proposé une caractérisation des antécédents des règles minimales non redondantes, appelées générateurs minimaux.

Définition 2 (Générateur Minimal) *Étant donné un itemset fermé X dans une base de données transactionnelles $\mathcal{D}$, un itemset $X' \subseteq X$ est un générateur minimal de X ssi $Supp(X', \mathcal{D}) = Supp(X, \mathcal{D})$ et il n'y a pas de $X'' \subseteq X$ tel que $X'' \subset X'$ et $Supp(X'', \mathcal{D}) = Supp(X, \mathcal{D})$.*

Les algorithmes usuels utilisent l'ensemble des itemsets fréquents et fermés avec les générateurs minimaux pour extraire l'ensemble des règles minimales non redondantes. Alors, les approches existantes pour la fouille des règles minimales procèdent en deux étapes. Dans notre approche, on propose d'étendre l'encodage SAT proposé dans [3] pour extraire les règles d'association minimales non redondantes en une seul étape.

Afin de définir un encodage SAT du problème d'énumération des règles d'association minimales non redondantes, il suffit d'étendre l'encodage décrit dans la Section 3 avec la formule qui oblige chaque antécédent à être un générateur minimal. À cette fin, nous utilisons une formule qui représente le fait que si $Supp(X \rightarrow Y, \mathcal{D}) = Supp(X \setminus \{a\} \rightarrow Y, \mathcal{D})$, alors a doit être exclus de X c-à-d $a \notin X$. Cependant, Nous écrivons la contraposition de cette propriété. En effet, la formule (8) exprime que pour tout item a, si a appartient à X alors le support de X est inférieur à celui de $X \setminus \{a\}$.

On utilise $\mathcal{E}_{MNR}(\mathcal{D}, minsupp, minconf)$ pour noter l'encodage $(1) \wedge (2) \wedge (3) \wedge (4) \wedge (5) \wedge (6) \wedge (7) \wedge (8)$.

La validité de $\mathcal{E}_{MNR}(\mathcal{D}, minsupp, minconf)$ vient directement de la proposition suivante :

Proposition 2 *La règle d'association $r : X \rightarrow Y$ est une règle minimale non redondante ssi r est une règle d'association fermée, et $|X| = 1$ ou, pour tout item $a \in X$, $Supp(X, \mathcal{D}) < Supp(X \setminus \{a\}, \mathcal{D})$.*

Preuve 2
Partie $\Rightarrow$. *En utilisant la proposition 1, nous savons que r est une règle d'association fermée. Supposons qu'il existe un item $a \in X$ tel que $Supp(X, \mathcal{D}) = Supp(X \setminus \{a\}, \mathcal{D})$. Alors, $r' : X \setminus \{a\} \rightarrow Y \cup \{a\}$ est une règle d'association fermée telle que $Supp(r, \mathcal{D}) = Supp(r', \mathcal{D})$ et $Conf(r, \mathcal{D}) = Conf(r', \mathcal{D})$. Ainsi, nous avons une contradiction puisque r est une règle d'association minimale non redondante.*
Partie $\Leftarrow$. *En utilisant le fait que r est une règle d'association fermée, nous savons qu'il n'y a pas de règle d'association $r' : X' \rightarrow Y'$ telle que $X \cup Y \subset X' \cup Y'$ et $Supp(r, \mathcal{D}) = Supp(r', \mathcal{D})$. De plus, sachant que $Supp(X, \mathcal{D}) < Supp(X \setminus \{a\}, \mathcal{D})$ pour chaque $a \in X$, nous obtenons $Conf(X \setminus \{a\} \rightarrow Y \cup \{a\}, \mathcal{D}) < Conf(r, \mathcal{D})$ pour chaque $a \in X$. En conséquence, r est une règle d'association non redondante.*

$$(\bigwedge_{a\in\Omega} x_a \rightarrow \bigvee_{(i\in 1..m,\ a\notin I_i)} (\bigwedge_{b\notin I_i\cup\{a\}} \neg x_b)) \vee (\sum_{b\in\Omega} x_b = 1) \quad (8)$$

$$(\bigwedge_{a\in\Omega} (x_a \rightarrow \bigvee_{(i\in 1..m,\ a\notin I_i)} \neg z_i)) \wedge (\bigwedge_{i\in 1..m} (\neg z_i \rightarrow \sum_{b\notin I_i} x_b \leq 1)) \vee (\sum_{b\in\Omega} x_b = 1) \quad (9)$$

$$(\neg x_1 \vee s_1) \wedge (\neg x_n \vee \neg s_{n-1}) \wedge \bigwedge_{1<i<n} (\neg x_i \vee s_i) \wedge (\neg s_{i-1} \vee s_i) \wedge (\neg x_i \vee \neg s_{i-1}) \quad (10)$$

$$(\neg x_1 \vee s_1) \wedge (y \vee \neg x_n \vee \neg s_{n-1}) \wedge \bigwedge_{1<i<n} (\neg x_i \vee s_i) \wedge (\neg s_{i-1} \vee s_i) \wedge (y \vee \neg x_i \vee \neg s_{i-1}) \quad (11)$$

La validité de notre encodage signifie qu'une interprétation booléenne $\mathcal{I}$ est un modèle de $\mathcal{E}_{MNR}(\mathcal{D}, minsupp, minconf)$ si et seulement si $X = \{a \in \Omega \mid \mathcal{I}(x_a) = 1\} \rightarrow Y = \{b \in \Omega \mid \mathcal{I}(y_b) = 1\}$ est une règle d'association minimale non redondante.

Proposition 3 *L'encodage* $\mathcal{E}_{MNR}(\mathcal{D}, minsupp, minconf)$ *est valide.*

Preuve 3 *ça provient de la validité de l'encodage* (1) ∧ (2) ∧ (3) ∧ (4) ∧ (5) ∧ (6) ∧ (7) *en respectant le problème de génération des règles d'association fermées, la proposition 2 et le fait que* (8) *exprime que* $Supp(X, \mathcal{D}) < Supp(X \setminus \{a\}, \mathcal{D})$ *pour chaque* $a \in X$.

Notons que la contrainte (8) n'est pas une formule CNF. Afin d'éviter l'explosion en terme de nombre de clauses résultantes de la transformation de (8) en CNF, des nouvelles variables peuvent être ajoutées pour représenter les sous formules de la forme $\bigwedge_{b\notin I_i\cup\{a\}} \neg x_b$ i.e., $z_i \leftrightarrow \bigwedge_{b\notin I_i\cup\{a\}} \neg x_b$. Toutefois, en utilisant cette transformation, le nombre de clauses résultantes est en $O(m \times |\Omega|^2)$ ce qui peut rendre l'énumération des modèles beaucoup plus difficile. Pour limiter le nombre de clauses, nous proposons la transformation exprimée dans la formule (9) qui est équivalente à la propriété capturée par (8).

En effet, cette transformation vient du fait que $(\bigwedge_{b\notin I_i\cup\{a\}} \neg x_b)$ est équivalente à $(\sum_{b\notin I_i} x_b \leq 1)$ dans le cas où I_i ne contient pas a. En conséquence, (9) exprime exactement les exigences de (8). Les variables additionnelles z_i permettent d'obtenir un encodage efficace.

Notons que (9) peut être encodée en $O(m \times |\Omega|)$ plutôt que $O(m \times |\Omega|^2)$ de la formulation précédente. Une contrainte linéaire de la forme $\sum_{i=1}^{n} x_i \leq 1$ peut être encodée linéairement [16] en utilisant des variables additionnelles comme le montre la formule (10).

Ainsi, la contrainte $(\neg y \rightarrow \sum_{i=1}^{n} x_i \leq 1)$ est encodée en rajoutant y à chaque clause de (10). Cependant, cela peut ralentir le processus de la propagation unitaire. En effet, lorsque plus qu'une variable x_i est affectée à *vrai*, y ne se déduit pas pour être vrai directement par propagation unitaire. Pour augmenter la puissance de la propagation unitaire, il faut ajouter y uniquement au clauses binaires négatives de (10) comme le montre la formule (11).

Il convient de noter qu'on peut utiliser certaines des contraintes ci-dessus pour énumérer tous les générateurs minimaux. Comme mentionné précédemment, les générateurs minimaux sont les antécédents des règles minimales non redondantes. En conséquence, l'encodage (3) ∧ (5) ∧ (9) (limité à X) permet d'avoir tous ces générateurs minimaux.

Une autre notion de règles non redondantes a été définie dans le travail de M. Zaki [18]. C'est légèrement différent des règles représentatives définies dans [13]. Ce problème consiste à extraire des règles d'association , appelées les règles les plus générales (MGR), qui ont le plus petit antécédent et le plus petit conséquent (en termes d'inclusion) dans une classe d'équivalence de règles (avec le même support et la même confiance)

Définition 3 *[18] Une règle d'association* $r : X \rightarrow Y$ *est une règle non redondante ssi il n'y a pas de règle* $r' : X' \rightarrow Y'$ *différente de* r *tels que (i)* $Supp(r) = Supp(r')$*, (ii)* $Conf(r) = Conf(r')$ *et (iii)* $X' \subseteq X$ *et* $Y' \subseteq Y$.

Contrairement à la notion de non redondance exprimée par la définition 1, la contrainte de fermeture sur $X \cup Y$ est évidemment omise par la définition de Zaki.

Exemple 3 *Considérant à nouveau les règles d'association de la Table 2.* $r_2 : \{a\} \rightarrow \{b, c, d\}$ *est redondante tandis que* $r_1 : \{a\} \rightarrow \{b\}$ *ne l'est pas.*

La proposition 4 donne une caractérisation des règles d'association non redondantes de Zaki.

Proposition 4 *Etant donnée une règle d'association* $r : X \rightarrow Y$ *dans une base de données transactionnelles*

$\mathcal{D}$, r est une règle non redondante ssi (i) $|X| = 1$ ou $\forall a \in X$, $Supp(X \setminus \{a\}, \mathcal{D}) > Supp(X, \mathcal{D})$; et (ii) $|Y| = 1$ ou $\forall b \in Y$, $Supp(X \cup Y) < Supp(X \cup Y \setminus \{b\})$.

Preuve 4
Partie $\Rightarrow$. *Supposons que $|X| > 1$ et qu'il existe $a \in X$ tel que $Supp(X \setminus \{a\}, \mathcal{D}) = Supp(X, \mathcal{D})$. Alors, $Supp(X \setminus \{a\} \rightarrow Y, \mathcal{D}) = Supp(r, \mathcal{D})$ tient. De plus, nous avons $Supp(X \cup Y \setminus \{a\}, \mathcal{D}) = Supp(X \cup Y, \mathcal{D})$. Ainsi, nous avons $Conf(X \setminus \{a\} \rightarrow Y, \mathcal{D}) = Conf(r, \mathcal{D})$. En conséquence, nous avons une contradiction puisque r est une règle non redondante et nous obtenons la propriété (i).*

Supposons maintenant qu'il existe $b \in Y$ tel que $|Y| > 1$ et $Supp(X \cup Y \setminus \{b\}, \mathcal{D}) = Supp(X \cup Y, \mathcal{D})$. Alors, $Conf(X \rightarrow Y \setminus \{b\}, \mathcal{D}) = Conf(X \rightarrow Y, \mathcal{D})$ tient. De plus, nous avons $Supp(X \rightarrow Y \setminus \{b\}, \mathcal{D}) = Supp(X \rightarrow Y, \mathcal{D})$. Ainsi, en utilisant le fait que r est une règle non redondante, nous avons une contradiction et nous obtenons la propriété (ii).

Partie $\Leftarrow$. *Supposons que r est une règle non redondante. Alors , il existe $a \in X \cup Y$ tel que $Supp(X \setminus \{a\} \rightarrow Y, \mathcal{D}) = Supp(r, \mathcal{D})$ si $a \in x$, et $Conf(X \rightarrow Y \setminus \{a\}, \mathcal{D}) = conf(r, \mathcal{D})$ sinon. Ainsi, nous avons $Supp(X \setminus \{a\}, \mathcal{D}) = Supp(X, \mathcal{D})$ si $a \in X$, et $Supp(X \cup Y) = Supp(X \cup Y \setminus \{b\})$ sinon. En conséquence, en utilisant les propriétés (i) et (ii) nous avons une contradiction. Donc, r est non redondante.*

En utilisant la caractérisation donnée par la Proposition 4, il suffit d'ajouter à l'encodage $\mathcal{E}_{MNR}(\mathcal{D}, minsupp, minconf)$, sans contrainte de fermeture, la nouvelle contrainte (12) qui représente la propriété (ii) pour avoir un encodage qui correspond au règles non redondante de Zaki.

Il convient de noter que la contrainte (12) est très similaire à (8). La différence réside dans le fait que nous utilisons les variables p_i pour résonner sur la couverture de $X \cup Y$ et non pas Y. En outre, on peut facilement voir que (12) peut être encoder en une formule CNF de la même façon que (8).

5 Expérimentations

Dans cette section, nous présentons une évaluation expérimentale comparative de l'approche proposée avec des algorithmes spécialisés dans l'extraction des règles d'association. Nous considérons la tâche de fouille de règles d'association minimales non redondantes (MNR).

Pour énumérer l'ensemble des modèles de la formule CNF résultante, nous suivons l'approche de [3]. L'algorithme d'énumération des modèles proposé est basé sur une procédure de recherche DPLL. Dans nos expérimentations, l'heuristique de choix de variable se concentre en priorité sur les variables X et Y respectivement pour sélectionner celle qui sera affecté par la suite. La puissance de cette approche consiste à utiliser la structure des littéraux observés pour effectuer efficacement la propagation unitaire. Notons également que les contraintes (5) et (6), dédiées à la fréquence et à la confiance, sont gérées dynamiquement sans transformation en forme CNF conduisant à un algorithme d'énumération de modèles hybride SAT-CSP. En effet, les inégalités linéaires (5) et (6) Sont gérées et propagées à la volée, comme se fait généralement en programmation par contraintes. Chaque modèle de la formule propositionnelle, encodant la tâche de fouille de règles d'association, correspond exactement à une règle d'association en considérant la valeur de vérité des variables propositionnelles qui encodent l'antécédent (X) et le conséquent (Y) de cette règle.

Dans ces expérimentations, *SAT4MNR* indique notre solveur basé sur SAT pour la fouille des règles d'association minimales non redondantes. De plus, nous considérons *SAT4MNR-D* qui partitionne la recherche comme dans [10]. Cela se fait comme suit : Soit $\Omega = \{a_1, \ldots, a_n\}$, on transforme le problème en n problème de fouille où chacun encode les règles $X \rightarrow Y$ tel que $\{a_1 \ldots, a_{i-1}\} \not\subset X$ et $a_i \in X$. On note aussi *SAT4MGR* notre solveur basé sur SAT pour la fouille des règles les plus générales (Définition 3).

Afin d'évaluer la performance de notre approche SAT dédiée à la fouille des règles minimales non redondantes, Nous comparons notre solveur à deux solveurs spécialisés dans l'extraction de règles d'association, à savoir *CORON*[1] et *SPMF*[2] [4]. *CORON* et *SPMF* sont deux outils implémentés en Java qui intègrent une riche collection d'algorithmes de fouille de données. Pour les règles d'association *minimales non redondantes*, nous comparons notre approche à l'algorithme *ZART* implémenté dans *CORON* et *SPMF*. *ZART* est l'un des plus récents et efficaces algorithmes de l'état de l'art pour l'énumération des règles d'association minimales non redondantes [17]. Rappelons que *ZART* trouve les règles minimales non redondantes en deux étapes. Tout d'abord, l'ensemble de tous les itemset fermés et fréquents ainsi que les générateurs minimaux sont extraits rapidement. Ensuite, L'identification des règles non redondantes est effectuée. Cette procédure à deux étapes consomme plus de temps.

Pour évaluer les performances de l'approche proposée, on varie le support et la confiance de 5% à 100% avec un intervalle de taille 5. Ainsi, un ensemble de 400 configurations est généré pour chaque jeu de don-

1. Coron : http ://coron.loria.fr/site/system.php
2. SPMF : http ://www.philippe-fournier-viger.com/spmf/

$$\bigwedge_{a\in\Omega} y_a \rightarrow (\bigvee_{(i\in 1..m,\ a\notin I_i)} (p_i \wedge \bigwedge_{b\notin I_i\cup\{a\}} \neg y_b)) \vee (\sum_{b\in\Omega} y_b = 1) \quad (12)$$

nées . Toutes les expérimentations ont été effectuées sur une machine Intel Xeon quad-core avec 32GB de RAM et une fréquence de 2.66 Ghz. On fixe une limite de temps de 15 minutes de temps CPU pour chaque instance (configuration).

Résultats : La Table 3 décrit nos résultats comparatifs. Nous rapportons dans la colonne 1 les nom des jeux de données et leurs caractéristiques entre parenthèses : nombre d'items (#items), nombre de transactions (#trans) et la densité. Pour chaque algorithme, on rapporte le nombre de configurations résolues ($\#S$), et le temps moyen de résolution (*avg.time* en secondes). Pour chaque instance non résolue, le temps est mis à 900s (limite de temps). Dans la dernière ligne de la Table 3, on donne le nombre d'instances résolues et le temps CPU moyen global en secondes.

Selon ces résultats, *SAT4MNR* surpasse les deux solveurs spécialisés *CORON* et *SPMF*. Il résout 488 configurations de plus que *CORON* et 920 de plus que *SPMF*. *SAT4MNR-D* est le meilleur sur tous les jeux de données en termes de nombre de configurations résolues et en temps CPU moyen, à l'exception de *mushroom* où *CORON* est meilleur en temps mais *SAT4MNR-D* résout toutes les configuration. Remarquons que pour *mushroom*, le nombre de règles d'association minimales non redondantes est très limité. Cela explique pourquoi *SAT4MNR* n'est pas meilleur que *CORON* sur ce jeu de données. Par exemple, pour le jeu de données *anneal*, *SAT4MNR* est remarquablement efficace. Il résout environ 100 configurations de plus que *CORON* et environ 200 configurations de plus que *SPMF*. On peut aussi remarquer que pour le jeu de données *Lymph*, *SAT4MNR-D* résout toutes les configurations en un temps moyen de $7s$ tandis que *CORON* et *SPMF* ne peuvent pas résoudre toute les configurations et ils prennent beaucoup de temps comparés à *SAT4MNR-D*. Plus généralement, plus la densité des données est élevée, meilleures sont les performances de *SAT4MNR*. Il est intéressant de noter que le partitionnement du problème de fouille permet de pousser davantage les performances de *SAT4MNR*. En effet, *SAT4MNR-D* permet d'obtenir de meilleures performances c-à-d 168 instances résolues et le temps de résolution moyen est améliorée de 235.15 à 215.31. *SAT4MGR* résout moins de configurations que *SAT4MNR* car l'ensemble de règles minimales non redondantes est connu pour être plus réduit comparé à celui des règles les plus générales non redondantes.

La Figure 1 représente le comportement de l'approche considérée pour l'extraction de règles d'association sur deux jeux de données représentatifs *Anneal* et $kr-vs-kp$. Les résultats sont obtenus en variant un paramètre, tout en maintenant les autres fixés. Lorsque le seuil minimum de support diminue, le temps nécessaire pour trouver toutes les règles augmente. Remarquons que pour *CORON* et *SPMF* le temps augmente rapidement comparé à *SAT4MNR-D*. Pour *anneal*, *SPMF* (resp. *CORON*) n'est pas capable d'extraire toutes les règles non redondantes lorsque le seuil minimum de support est inférieur à 85%(resp. 65%). Par contre, avec *SAT4MNR* et *SAT4MNR-D* il est possible d'obtenir toutes les règles pour toutes les valeurs du seuil minimum de support. Pour *kr-vs-kp*, Il est important de noter que le temps nécessaire pour extraire les règles augmente considérablement pour *SPMF* et *CORON* même pour une confiance élevée. Par exemple, quand le support minimum va de 100% à 80% le temps est multiplié au moins par un facteur de 10. Une telle augmentation est très limitée pour *SAT4MNR* et *SAT4MNR-D*.

Finalement, la Table 4 présente la variation du rapport entre le nombre de règles classiques (pures), les règles fermées, les règles non redondantes les plus générales et les règles minimales non redondantes pour le jeu de données *kr-vs-kp*. Comme on peut l'observer, le nombre de règles minimales non redondantes est plus petit que celui des règles non redondantes les plus générales. Ce dernier est plus petit que le nombre des règles d'association fermée, qui est en soi plus petit que celui des pures. Par exemple, lorsque le seuil minimum de support est égal à 40%, les règles d'association minimales non redondantes présentent 2.85% de toutes les règles d'association classiques tandis que les règles non redondantes les plus générales représentent environ 3.90%.

6 Conclusion et perspectives

Dans cet article, nous avons proposé une nouvelle approche pour la découverte des règles d'association minimales non redondantes. On a montré que les règles non redondantes avec un minimum d'antécédents et un maximum de conséquences peuvent être capturées en modélisant ce problème en un problème de satisfaibilité propositionnelle (SAT). On a démontré que notre approche est déclarative et flexible. En effet, on a montré que les générateurs minimaux peuvent être extraits en utilisant des contraintes similaires. On a également

	SAT4MNR-D		*SAT4MNR*		*CORON*		*SPMF*		*SAT4MGR*	
jeux de données (#items, #trans, density)	#S	avg. time(s)	#S	avg. time(s)	#S	avg. time(s)	#S	avg. time(s)	#S	avg. time(s)
Audiology (148, 216, 45%)	21	854,82	21	854.87	20	855.01	20	855.00	20	855.00
Zoo-1 (36, 101, 44%)	400	0.23	400	0.27	400	1.35	373	108.60	400	0.71
Tic-tac-toe (27, 958, 33%)	400	0.34	400	0.14	400	0.24	400	0.20	400	0.61
Anneal (93, 812, 45%)	279	337.25	248	405.82	160	591.39	80	724.46	221	461.05
Australian-c (125, 653, 41%)	298	265.74	278	309.32	251	352.01	220	417.94	263	358.40
German-c (112, 1000, 34%)	354	149.03	328	212.58	321	206.34	278	294.45	304	272.88
H-cleveland (95, 296, 47%)	331	200.28	317	235.79	271	307.57	240	368.21	286	289.28
Hepatitis (68, 137, 50%)	360	140.69	343	170.89	286	284.09	260	331.57	315	228.13
Hypothyroid (88, 3247, 49%)	150	615.13	126	649.22	104	681.52	80	751.23	109	676.03
kr-vs-kp (73, 3196, 49%)	198	504.62	172	556.85	168	552.04	140	627.64	158	583.25
Lymph (68, 148, 40%)	400	6.78	400	19.21	357	131.07	280	316.78	395	37.15
Mushroom (119, 8124, 18%)	400	146.87	389	77.02	400	3.81	360	97.25	354	181.89
P-tumor (31, 336, 48%)	400	2.08	400	4.61	400	4.15	379	87.66	400	8.11
Soybean (50, 650, 32%)	400	0.36	400	0.20	400	0.61	380	48.51	400	2.26
Vote (48, 435, 33%)	400	5.43	400	30.46	364	87.56	380	84.82	372	111.06
Total	**4790**	**215.31**	**4622**	**235.15**	4302	270.58	3870	340.94	4397	271.05

TABLE 3 – Règles d'Association Non Redondantes : *SAT4MNR* vs *CORON* vs *SPMF*

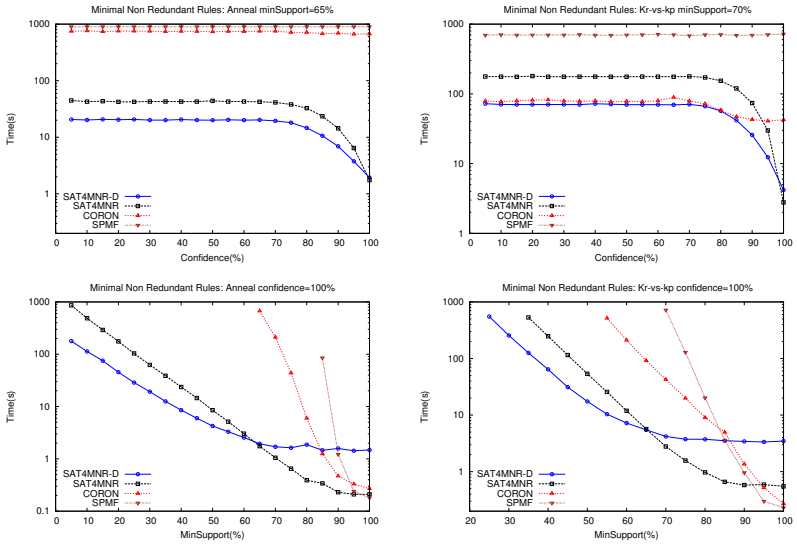


FIGURE 1 – Résultats des testes : *Anneal* et *kr-vs-kp*

montré comment capturer les règles non redondantes avec un minimum d'antécédents et un maximum de conséquences. L'évaluation expérimentale montre que Notre approche proposée offre de meilleures performances que les techniques de fouille spécialisées.

Comme perspectives, on envisage aborder la question de l'extraction des règles les plus générales ayant des itemsets adjacents [18] en utilisant la satisfiabilité pour avoir une représentation compacte de l'ensemble des règles non redondante les plus générales.

Références

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of SIGMOD'93*, pages 207–216, 1993.

[2] Yves Bastide, Nicolas Pasquier, Rafik Taouil, Gerd Stumme, and Lotfi Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *Computational Logic - CL 2000*, volume 1861, pages 972–986, 2000.

[3] Abdelhamid Boudane, Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. A sat-based approach for mining association rules. In *Proceedings of IJCAI'16*, pages 2472–2478, 2016.

[4] Philippe Fournier-Viger, Antonio Gomaric, Ted Gueniche, Azadeh Soltani, Cheng-Wei Wu, and Vincent S Tseng. Spmf : a java open-source pattern mining library. *The Journal of Machine Learning Research*, 15(1) :3389–3393, 2014.

seuil minimum de support (%)	40	45	50	55	60	65	70
$\#Pures/\#Closed$	7.67	5.68	3.64	2.99	2.46	1.95	1.67
$\#Closed/\#MGR$	2.40	2.16	1.95	1.78	1.61	1.46	1.35
$\#MGR/\#MNR$	1.94	1.83	1.73	1.63	1.54	1.45	1.38

TABLE 4 – *kr-vs-kp* : Pure vs Closed vs MNR vs MGR

[5] Philippe Fournier-Viger and Vincent S. Tseng. TNS : mining top-k non-redundant sequential rules. In *Proceedings of SAC '13*, pages 164–166, 2013.

[6] Ghada Gasmi, Sadok Ben Yahia, Engelbert Mephu Nguifo, and Yahya Slimani. IGB : A new informative generic base of association rules. In *Proceedings of PAKDD'05*, pages 81–90, 2005.

[7] Tias Guns, Siegfried Nijssen, and Luc De Raedt. Itemset mining : A constraint programming perspective. *Artif. Intell.*, 175(12-13) :1951–1983, 2011.

[8] Tias Guns, Siegfried Nijssen, and Luc De Raedt. k-pattern set mining under constraints. *IEEE Trans. Knowl. Data Eng.*, 25 :402–418, 2013.

[9] Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. The top-k frequent closed itemset mining using top-k sat problem. In *Proceedings of ECML/PKDD'13*, pages 403–418, 2013.

[10] Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. Decomposition based SAT encodings for itemset mining problems. In *Proceedings of PAKDD'15*, pages 662–674, 2015.

[11] Matti Järvisalo. Itemset mining as a challenge application for answer set enumeration. In *LPMNR*, pages 304–310. Springer, 2011.

[12] Mehdi Khiari, Patrice Boizumault, and Bruno Crémilleux. Constraint programming for mining n-ary patterns. In *In Proceedings of CP'10*, pages 552–567. Springer, 2010.

[13] Marzena Kryszkiewicz. Representative association rules. In *Proceedings of PAKDD'98*, pages 198–209, 1998.

[14] Marzena Kryszkiewicz. Representative association rules and minimum condition maximum consequence association rules. In *Proceedings of PKDD '98*, pages 361–369, 1998.

[15] Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni. A constraint language for declarative pattern discovery. In *Proceedings of SAC'12*, pages 119–125, 2012.

[16] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proceedings of CP'05*, pages 827–831, 2005.

[17] Laszlo Szathmary, Amedeo Napoli, and Sergei O. Kuznetsov. ZART : A multifunctional itemset mining algorithm. In *Proceedings of ICCLTA'07*, 2007.

[18] Mohammed Javeed Zaki. Mining non-redundant association rules. *Data Mining Knowledge Discovery*, 9 :223–248, 2004.

Caractérisation de nouvelles classes traitables en SAT via la théorie des graphes

Yazid M. Boumarafi * **Lakhdar Sais Yakoub Salhi**

CRIL - CNRS, Artois University Lens France

{boumarafi,sais,salhi}@cril.fr

Résumé

Dans ce papier, nous proposons une nouvelle approche permettant de définir de nouvelles classes traitables pour le problème de la satisfiabilité propositionnelle (SAT). L'idée de base consiste à transformer une instance SAT en une instance du problème de recherche d'un stable maximum dans un graphe. Dans ce contexte, et sans perte de généralité, nous considérons les formules propositionnelles mises sous forme normale conjonctive, où les clauses sont soit positives, soit binaires négatives. Des classes traitables sont ensuite dérivées de l'existence d'un algorithme polynomial pour le problème de recherche d'un stable maximum pour certaines classes de graphes comme les graphes sans étoiles (claw-free) et les graphes parfaits (perfect graphs). Nous montrons, en particulier, que le problème bien connu des pigeons (Pigeon Hole problem) appartient à une de ces classes traitables. Finalement, nous proposons une nouvelle caractérisation des modèles minimaux dans le plus grand fragment considéré en se basant sur le problème de recherche d'un stable maximum.

Abstract

In this paper, we propose a new approach for defining tractable classes for the propositional satisfiability problem (in short SAT). The basic idea consists in transforming SAT instances into instances of the problem of finding a maximum independent set. In this context, we only consider propositional formulæ in conjunctive normal form where each clause is either positive or binary negative. Tractable classes are obtained from existing polynomial time algorithms for the problem of finding a maximum independent set in the case of different graph classes, such as claw-free graphs and perfect graphs. We show, in particular, that the pigeonhole problem belongs to one of the defined tractable classes. Furthermore, we propose a characterization of the minimal models in the largest considered fragment based on the maximum independent set problem.

*Papier doctorant : Yazid M. Boumarafi est auteur principal.

1 Introduction

Depuis plusieurs années le problème de la satisfiabilité des formules propositionnelles (SAT : décider si une formule propositionnelle sous forme normale conjonctive admet ou non un modèle) fait l'objet de plusieurs travaux très avancés sur les deux plans (pratique et théorique) en informatique, il est l'un des problèmes centraux dans la théorie de complexité et l'informatique en général. Sur le plan pratique, ces dernières années ont connues une percée remarquable dans la résolution du problème SAT, les solveurs modernes sont capables de résoudre différents types d'instances avec plusieurs centaines de milliers de variables et des millions de clauses. Sur le plan théorique, SAT est le premier problème à avoir été prouvé NP-complet [10]. Cela n'a pas empêché l'identification de plusieurs classes de formules propositionnelles pour lesquelles la résolution de SAT est polynomial. Ces classes sont appelées traitables ou polynomiales, parmi ces classes, nous citons : les formules de Horn [17], les formules Horn renommables [26], les formules Q-Horn [5], les formules Krom [25].

La caractérisation de nouvelles classes traitables est très importante pour SAT et aussi pour l'IA (Intelligence Artificielle). En effet, de nombreux problèmes en IA peuvent être réduits à SAT. Par exemple, plusieurs systèmes de représentation des connaissances sont basés sur les formules de Horn pour lesquelles SAT peut être résolu en temps linéaire. De plus, dans la compilation de connaissances des formules générales sont parfois approchées par des formules traitables (exemple [36, 4, 7]).

Nous pensons également que l'extension des fragments polynomiaux actuels de SAT pourrait constituer un pas en avant dans la compréhension de l'efficacité pratique des solveurs SAT. La caractérisation de nou-

velles classes traitables aiderait à mieux comprendre la traitabilité / intraitabilité des instances SAT.

En dépit du fait que les problèmes NP-complet connus admettent leurs propres classes traitables, il y a peu de travaux portés sur la recherche de leurs contreparties en SAT. Par exemple plusieurs algorithmes résolvent en temps polynomial des problèmes pour des graphes appartenant à des classes spéciales. Par exemple, plusieurs problèmes connus admettent des algorithmes polynomiaux pour les graphes parfaits [20], convexes et cordaux [15], sans AT [37] et sans étoiles [18] (voir [6] pour un résumé). Dans [31], les auteurs montrent que les formules CNF avec des hypergraphes β-acycliques peuvent être résolues en temps polynomial. On peut également mentionner plusieurs classes traitables récentes mises en évidence par différents auteurs dans les problèmes de satisfaction des contraintes (CSP) (voir par exemple [12]).

En exploitant la réduction polynomiale, l'un des outils fondamentaux de la théorie de la complexité, nous donnons une caractérisation de nouvelles classes traitables en SAT en exploitant les résultats obtenus par [30, 28] dans le problème de recherche d'un stable de taille maximum qui est traitable dans le cas de certaines classes de graphe (les graphes sans étoile). Sans perte de généralité, nous considérons une formule propositionnelle mise sous forme normale conjonctive (CNF) constituée uniquement d'un ensemble de clauses positives et d'un ensemble de clauses binaires négatives (PBN). Chaque formule CNF peut être transformée en une formule PBN en utilisant la transformation de Tseitin, nous pensons que les formules PBN sont plus adaptées pour établir des liens directs avec les problèmes en théorie des graphes et le problème de satisfaction de contraintes (CSP). Par exemple, l'encodage direct [38] (*direct encoding*) d'un CSP binaire en une CNF est une formule PBN. En prenant en compte les occurrences des variables dans l'ensemble des clauses positives, nous définissons une hiérarchie des formules propositionnelles. Notre approche de caractérisation de nouvelles classes traitables est basée sur la réduction du problème SAT de chaque fragment de la hiérarchie au problème de recherche d'un stable de taille maximum. Ceci nous permet de trouver les contreparties équivalentes de quelques classes traitables du problème de recherche d'un stable de taille maximum. Nous montrons aussi que le graphe correspondant à la formule CNF représentant le problème des pigeons appartient à une classe particulière de graphes, la classe des graphes sans étoile. En résumé, nous proposons une nouvelle approche permettant de définir de nouvelles classes traitables pour SAT, en transformant une instance SAT en une instance du problème de recherche d'un stable maximum. De plus, nous proposons une caractérisation des modèles minimaux dans le plus grand fragment de la hiérarchie considérée, basée sur le problème de recherche d'un stable maximum. Cette caractérisation est utilisée pour mettre en évidence une connexion avec le problème du calcul des modèles minimaux, central dans de nombreux systèmes de raisonnement en Intelligence Artificielle, comme la circonscription propositionnelle [29, 27], le diagnostic minimal [34, 16]. Cette dernière contribution nous permet de montrer la richesse des liens et connexions entre différents problèmes.

Ce papier est organisé de la manière suivante. Après quelques préliminaires sur la logique propositionnelle et le problème SAT, nous présentons notre hiérarchie des formules PBN. Une présentation incrémentale des classes traitables associées aux différents fragments de la hiérarchie est présentée en partant du fragment le plus spécifique. Nous décrivons ensuite notre caractérisation des modèles minimaux dans le plus grand fragment de la hiérarchie. Ensuite, nous discutons quelques travaux connexes, principalement ceux proposés dans le contexte des problèmes de satisfaction des contraintes avant de conclure.

2 Logique propositionnelle et problème SAT

Nous supposons que le lecteur connaît les principaux concepts de la logique propositionnelle. Ainsi, nous limitons notre présentation à certaines notations utilisées dans ce papier. Une formule propositionnelle en forme normale conjonctive (CNF) est une conjonction ($\wedge$) de clauses, une clause est une disjonction ($\vee$) de littéraux. Un littéral est une variable propositionnelle (p), appelée littéral positif, ou une variable propositionnelle négative ($\neg p$), appelée littéral négatif. Une clause est *binaire* si elle contient uniquement deux littéraux. Une clause est *positive* (resp. *négative*) si elle contient uniquement des littéraux positifs (resp. négatifs). Une formule CNF est un ensemble de clauses, une clause est un ensemble de littéraux. La *taille* d'une formule CNF ϕ correspond à $\sum_{c\in\phi} |c|$ où $|c|$ est le nombre de littéraux dans la clause c. Soit S un objet syntaxique, nous utilisons $\mathcal{P}(S)$ pour désigner l'ensemble des variables propositionnelles apparaissant dans S.
Une *interprétation booléenne* $\mathcal{I}$ d'une formule ϕ est une fonction allant de $\mathcal{P}(\phi)$ vers $\{0,1\}$ (0 correspond à *faux* et 1 correspond à *vrai*). Un *modèle* d'une formule ϕ est une interprétation booléenne $\mathcal{I}$ qui satisfait ϕ, i.e. $\mathcal{I}(\phi) = 1$. Une formule ϕ est dite consistante s'il existe un modèle pour ϕ. Le *problème SAT* consiste à décider si une formule CNF donnée admet ou non un modèle.
Soient c_i et c_j deux clauses contenant x et $\neg x$ respec-

tivement. Une *résolvante* de c_i et c_j sur x est définie par $r = c_i \cup c_j \backslash \{x, \neg x\}$. Notons que r est une conséquence logique de $c_i \wedge c_j$ i.e. chaque modèle de c_i et c_j est un modèle de r. La *règle de résolution* est définie comme le processus de génération d'une clause (appelée résolvante) à partir de deux clauses.

3 Formules PBN

Dans cette section, nous décrivons les fragments des formules propositionnelles que nous considérons.

Définition 1 (Formule PBN)**.** *Une formule ϕ en forme CNF est dite PBN si chacune de ses clauses est soit une clause positive ou une clause binaire négative. L'ensemble des clauses positives (resp. négatives) est noté Pos(ϕ) (resp. Nég(ϕ)).*

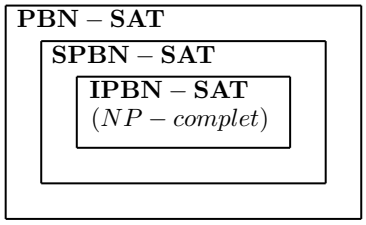


FIGURE 1 – Une hiérarchie des fragments syntaxiques de SAT

Pour transformer une formule CNF en une formule equi-satisfiable PBN, il existe plusieurs méthodes. Parmi ces méthodes, la transformation de Tseitin, en effet, il suffit d'associer pour chaque littéral négative l une variable propositionnelle notée r_l, et remplacer toute clause de la forme $p_1 \vee p_2 \vee ... \vee p_m \vee \neg q_1 \vee ... \vee \neg q_n$ par l'ensemble : $\{p_1 \vee p_2 \vee ... \vee p_m \vee r_{\neg q_1} \vee ... \vee r_{\neg q_n}, \neg q_1 \vee \neg r_{\neg q_1}, ..., \neg q_n \vee \neg r_{\neg q_n}\}$. De toute évidence, décider de la satisfiabilité d'une formule PBN reste NP-complet.

Maintenant, on définit deux types particuliers de formule PBN, en se basant sur les occurrences des littéraux positifs. Une formule SPBN est une formule PBN où chaque clause positive a au plus un littéral en commun avec toutes les clauses positives, or dans les formules IPBN, l'intersection des clauses positives donne l'ensemble vide (pas de littéraux en commun).

Définition 2 (Formule SPBN)**.** *Une formule SPBN est une formule PBN où, pour chaque $c \in Pos(\phi)$, $|\mathcal{P}(Pos(\phi) \setminus \{c\}) \cap c| \leqslant 1$*

Définition 3 (Formule IPBN)**.** *Une formule IPBN est une formule PBN où, pour chaque $c, c' \in Pos(\phi)$ avec $c \neq c'$, $c \cap c' = \emptyset$.*

Exemple 1. *La formule suivante est une formule SPBN : $(p \vee q) \wedge (p \vee r) \wedge (p \vee s) \wedge (t \vee u) \wedge (\neg q \vee \neg r) \wedge (\neg r \vee \neg s) \wedge (\neg p \vee \neg u)$. En effet, chaque clause positive a au plus un littéral en commun avec les clauses positives restantes.*

Considérant maintenant le problème des pigeons, attestant qu'il est impossible de mettre b pigeons dans n pigeonniers avec $b > n$. On note php_n^b le problème des pigeons avec b pigeons dans n pigeonniers. Cook a prouvé que le problème des pigeons admet une preuve courte dans le système de preuve par résolution étendue [11]. Une preuve courte existe également en utilisant une résolution par symétrie [1, 3]. Le problème des pigeons php_n^b peut être exprimé en utilisant la formule IPBN suivante $\phi_{php_n^b}$:

$$\bigvee_{j=1}^{n} p_{ij}, 1 \leqslant i \leqslant b \quad (1)$$

$$\neg p_{ij} \vee \neg p_{kj}, 1 \leqslant i < k \leqslant b \text{ and } 1 \leqslant j \leqslant n \quad (2)$$

Où $p_{ij} = 1$ si et seulement si le pigeon i est dans le pigeonnier j.

En effet, chaque clause positive de $\phi_{php_n^b}$ n'a aucun littéral en commun avec les autres clauses positives. Dans ce qui suit, nous montrons que décider de la satisfiabilité de la formule $\phi_{php_n^b}$ appartient à une classe traitable que nous définirons.

Pour faciliter la notation, nous utilisons PBN-SAT, SPBN-SAT et IPBN-SAT pour désigner le problème SAT dans le cas d'une formule PBN, une formule SPBN et une formule IPBN respectivement.

Proposition 1 (IPBN-SAT)**.** *Le problème IPBN-SAT est NP-complet.*

Preuve. Vu que le problème SAT est dans la classe NP, alors IPBN-SAT l'est aussi. Nous montrons maintenant que IPBN-SAT est NP-difficile, en utilisant le problème de coloration d'un graphe.

Il est connu que le problème de décider si un graphe admet une 3-coloration est un problème NP-difficile. Soit $G = (V, E)$ un graphe non orienté tel que V représente l'ensemble de ses sommets et E l'ensemble de ses arêtes. Une 3-coloration du graphe G correspond à une partition de l'ensemble V en 3 ensembles $P = \{V_{c1}, V_{c2}, V_{c3}\}$ de tel sorte que deux sommets dans le même sous ensemble ne sont pas adjacents, et V_{c1}, V_{c2}, et V_{c3} font référence à trois couleurs différentes c_1, c_2 et c_3 respectivement. Pour encoder le problème 3-coloration d'un graphe en une formule IPBN, on associe une variable propositionnelle $p_v^{c_i}$ à chaque sommet v coloré avec la couleur c_i. On a $\mathcal{I}(p_v^{c_i}) = 1$,

si le sommet v est coloré avec c_i et $\mathcal{I}(p_v^{c_i}) = 0$ sinon. Donc, le codage est défini comme suit :

$$\bigwedge_{v \in V} p_v^{c_1} \vee p_v^{c_2} \vee p_v^{c_3} \quad (3)$$

$$\bigwedge_{\{v,v'\} \in E} (\neg p_v^{c_1} \vee \neg p_{v'}^{c_1}) \wedge (\neg p_v^{c_2} \vee \neg p_{v'}^{c_2}) \wedge (\neg p_v^{c_3} \vee \neg p_{v'}^{c_3}) \quad (4)$$

Clairement, la formule (3) est un ensemble de clauses positives indépendantes, et par conséquent, (3) $\wedge$ (4) est une formule IPBN. Donc IPBN-SAT est un problème NP-complet. □

Il est aussi intéressant de noter que la NP-complétude d'un problème IPBN-SAT peut également être obtenue à partir des problèmes de satisfaction des contraintes. En effet, l'encodage direct [38] de tout CSP binaire en une formule CNF est une formule IPBN.

Un littéral *pure* est un littéral dont le littéral opposé n'apparait pas dans la formule CNF, la *règle des littéraux pures* consiste à supprimer les clauses contenant les littéraux pures dans la formule CNF. Intuitivement, une formule CNF est satisfiable si et seulement si la formule CNF obtenue après l'application de la règle des littéraux pures est satisfiable. A partir de maintenant, nous considérons seulement les formules PBN ne contenant aucun littéral pure.

Pour satisfaire une formule propositionnelle, il suffit de satisfaire un littéral dans chaque clause. Soit ϕ une formule PBN, on utilise $\mathcal{R}(\phi)$ pour noter la formule PBN $\phi \cup \{\neg p \vee \neg q \mid p \neq q$ et $\exists c \in Pos(\phi)$ tel que $\{p, q\} \subseteq c\}$. Nous montrons dans la proposition suivante que nous avons besoin de satisfaire seulement un littéral dans chaque clause positive.

Proposition 2. *Soit ϕ une formule SPBN, ϕ est satisfiable si et seulement si $\mathcal{R}(\phi)$ est satisfiable.*

Preuve. Vu que ϕ est un sous ensemble de $\mathcal{R}(\phi)$, si $\mathcal{R}(\phi)$ est satisfiable alors ϕ est satisfiable. Supposant maintenant que ϕ est satisfiable, et soit $\mathcal{I}$ un modèle de ϕ. Pour une clause positive $c \in Pos(\phi)$, $\ell_{c,\mathcal{I}}$ désigne un littéral positif dans c tel que (i) $\mathcal{I}(\ell_{c,\mathcal{I}}) = 1$ et (ii) si $|\mathcal{P}(\phi \setminus \{c\}) \cap c| = 1$ et $\mathcal{I}(\mathcal{P}(\phi \setminus \{c\}) \cap c) = 1$ alors $\ell_{c,\mathcal{I}} = \mathcal{P}(\phi \setminus \{c\}) \cap c$. Soit $\mathcal{I}'$ une interprétation booléenne de $\mathcal{R}(\phi)$ définie comme suit :

$$\mathcal{I}'(p) = \begin{cases} 1 & \text{si } \exists c \in Pos(\phi), p = \ell_{c,\mathcal{I}} \\ 0 & \text{sinon} \end{cases}$$

Il est évident que $\mathcal{I}'$ satisfait exactement un littéral dans chaque clause positive de ϕ, en effet, supposant qu'il existe une clause positive contenant deux littéraux distincts p et q tel que $\mathcal{I}'(p) = \mathcal{I}'(q) = 1$. Donc c partage les littéraux p et q avec d'autres clauses positives. Contradiction, car ϕ est une formule SPBN. Par conséquent, $\mathcal{I}'$ est un modèle de $\mathcal{R}(\phi)$. □

Nous présentons nos résultats de la maniére suivante, d'abord nous décrivons une approche de caractérisation des classes traitables pour IPBN-SAT, en suite nous généraliserons ces classes pour SPBN-SAT et enfin nous généraliserons notre approche pour la caractérisation des classes traitables pour PBN-SAT.

4 Classes traitables pour IPBN-SAT

Dans cette section, nous définissons une classe traitable pour les formules IPBN-SAT. Notre approche consiste en la transformation d'une instance de IPBN-SAT en une instances du problème de recherche d'un stable de taille maximum. Ce dernier admet une solution en un temps polynomial pour certaines classes de graphe. En particulier, on montre que l'instance représentant le problème des pigeons appartient à une classe traitable définie par cette approche. Soit ϕ une formule IPBN, $G_\phi^{\mathcal{P}}$ dénote le graphe non orienté correspondant, où $\{p, q\} \in E$ si et seulement si les variables p et q appartiennent à la même clause dans ϕ, i.e., $\exists c \in \phi$ tel que $\{p, q\} \subseteq \mathcal{P}(c)$. A titre d'exemple, le graphe correspondant à la formule suivante est représenté dans la figure 2 : $(p \vee q \vee r) \wedge (s \vee t) \wedge (\neg q \vee \neg s) \wedge (\neg r \vee \neg s) \wedge (\neg r \vee \neg t)$

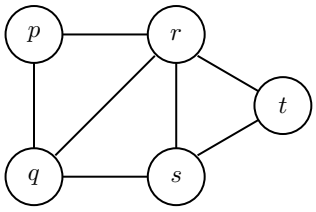


FIGURE 2 – Un graphe associé à une formule IPBN

Considérons un graphe non orienté $G = (V, E)$. Un stable est un sous-ensemble de sommets $S \subseteq V$ deux à deux non adjacents, i.e., pour chaque $v, v' \in S$ avec $v \neq v'$, $\{v, v'\} \notin E$. Un stable maximum de G est un sous-ensemble de sommets, le plus grand possible, tel que les éléments de ce sous-ensemble ne soient pas voisins. On utilise $\alpha(G)$ pour désigner un stable maximum de G. Le problème de recherche d'un stable maximum dans un graphe est un problème NP-difficile [6].

Par exemple, pour le graphe G représenté dans la figure 2 : $\alpha(G) = 2$ et ses stables maximums sont $\{p, s\}$, $\{p, t\}$ et $\{q, t\}$.

Théorème 1. *Soit ϕ une formule IPBN, ϕ est satisfiable si et seulement si $\alpha(G^{\mathcal{P}}_{\phi}) \geqslant n$ où $n = |Pos(\phi)|$.*

Preuve. **Partie** $\Rightarrow$. En utilisant la proposition 2, ϕ est satisfiable si et seulement si $\mathcal{R}(\phi)$ l'est. Soit $\mathcal{I}$ un modèle de $\mathcal{R}(\phi)$ et $S = \{p \in \mathcal{P}(\phi) \mid I(p) = 1\}$. Vu que $\mathcal{I}$ satisfait exactement un littéral dans chaque clause positive, $|S| \geqslant n$ est vérifié. Rappelons que les littéraux des clauses positives sont disjoints. Il suffit maintenant de montrer que S est un stable de $G^{\mathcal{P}}_{\phi}$. Supposant que $p, q \in S$ tel que $\{p, q\}$ est une arête dans $G^{\mathcal{P}}_{\phi}$, alors la clause $\neg p \vee \neg q$ appartient à $\mathcal{R}(\phi)$. Donc soit $\mathcal{I}(p) = 0$ ou bien $\mathcal{I}(q) = 0$, contradiction.

Partie $\Leftarrow$. Soit S un stable maximum de $G^{\mathcal{P}}_{\phi}$ tel que $|S| \geqslant n$ et soit $\mathcal{I}$ une interprétation booléenne de ϕ définie comme suit :

$$\mathcal{I}(p) = \begin{cases} 1 & \text{si } p \in S \\ 0 & \text{sinon} \end{cases}$$

En utilisant la définition de $G^{\mathcal{P}}_{\phi}$, $\mathcal{I}$ satisfait les clauses binaires négatives. En effet, pour chaque $\neg p \vee \neg q \in Nég(\phi)$, on a : $p \notin S$ ou bien $q \notin S$, car $\{p, q\}$ est une arête de $G^{\mathcal{P}}_{\phi}$. De plus, $\mathcal{I}$ satisfait au plus un littéral dans chaque clause positive, donc, vu que $S \subseteq \mathcal{P}(\phi)$ et $|S| \geqslant n$, $\mathcal{I}$ satisfait les clauses positives. □

Considérons à nouveau l'exemple précédent de la formule IBPN ϕ (son graphe corréspondant est représenté dans la figure 2). en utilisant le théorème 1, ϕ est satisfiable vu que le nombre de clauses positives est égal à 2 et $\alpha(G^{\mathcal{P}}_{\phi}) = 2$. Le stable maximum $\{p, s\}$ induit le modèle $\{p, s, \neg q, \neg r, \neg t\}$ de ϕ.

Plusieurs algorithmes de complexité polynomiale ont été proposés pour le problème de recherche d'un stable maximum pour certaines classes de graphe, comme les graphes sans étoile, graphes parfaits, graphes avec clique de largeur limitée [30, 21, 2, 6, 14]. Par conséquent, le théoréme 1 montre qu'on peut obtenir une classe traitable pour IPBN-SAT à partir du problème de recherche d'un stable de taille maximum. Considérons par exemple, la classe de graphes sans étoile. Un graphe $G = (V, E)$ est *sans étoile* si aucun sommet n'a trois voisins non adjacents par paire, i.e., pour chaque $v \in V$, s'il existe trois sommets distincts $v_1, v_2, v_3 \in V$ tel que $\{\{v, v_1\}, \{v, v_2\}, \{v, v_3\}\} \subseteq E$, alors au moins une de ces arêtes $\{v_1, v_2\}$, $\{v_2, v_3\}$ et $\{v_1, v_3\}$ appartient à E. Une *étoile* est un sous-graphe induit par quatre sommets où un des sommets a trois voisins non adjacents par paire. On dit qu'une formule IPBN ϕ est sans étoile si le graphe correspondant est sans étoile.

Théorème 2 (IPBN-SAT sans étoile)**.** *Le problème consistant à vérifier la satisfiabilité d'une formule IPBN sans étoile est traitable*

Preuve. Une conséquence directe du théorème 1 et le fait que le problème de recherche d'un stable de taille maximum dans la classe de graphes sans étoile est traitable. □

Il est évident que vérifier si un graphe non orienté est sans étoile ou non est traitable. On obtient le théorème suivant :

Théorème 3. *La vérification si une formule CNF est sans étoile est traitable.*

Considérant le problème de pigeons, dans la proposition suivante, nous montrons que vérifier la satisfiabilité de l'instance du problème des pigeons appartient à IPBN-SAT dont la formule CNF est sans étoile.

Proposition 3. *Le problème des pigeons est une instance de IPBN-SAT sans étoile.*

Preuve. Comme cité précédemment, la formule CNF $\phi_{php^b_n}$ représentant le problème des pigeons est une formule IPBN. Soit p, q, r, s différents sommets dans le graphe correspondant $G = (V, E)$ tel que $\{\{p, q\}, \{p, r\}, \{p, s\}\} \subseteq E$. Si deux des variables q, r, s appartiennent à la même clause positive, donc le sous graphe induit composé de ces quatre sommets n'est pas une étoile. Considérant maintenant le cas où q, r, s appartiennent à des clauses positives distinctes, alors deux variables parmi q, r et s représentent le même pigeonnier pour deux pigeons différents, vu que $\{\{p, q\}, \{p, r\}, \{p, s\}\} \subseteq E$. Par conséquent, une des clauses négatives suivantes : $\neg q \vee \neg r, \neg q \vee \neg s, \neg r \vee \neg s$ est dans $\phi_{PHP^b_n}$. Donc le sous graphe composé de ces quatre sommets n'est pas une étoile. □

Il est intéressant de noter qu'il existe des classes de graphe, autres que des graphes sans étoile, caractérisant des classes traitables dans le problème de recherche d'un stable maximum et qui peut être reconnu en temps polynomial. Par exemple, les graphes biparti, qui sont des graphes parfaits, peuvent être reconnus en temps polynomial.

5 Classes traitables pour SPBN-SAT

Dans cette section, une extension des résultats précédents est présentée afin de caractériser des classes traitables pour les formules SPBN. En effet, nous montrons que chaque instance de SPBN-SAT peut être traduite en une instance du problème de recherche d'un stable maximum. L'idée principale consiste à prendre en compte le nombre d'occurrences des littéraux en commun entre les clauses positives.

Soit ϕ une formule PBN et $p \in \mathcal{P}(\phi)$, on note par $Nb(p, \phi)$ la valeur $|\{c \in Pos(\phi) \mid p \in c\}|$, de plus, on

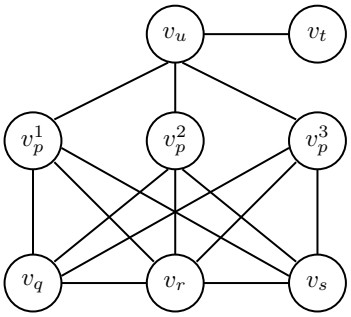


FIGURE 3 – Un graphe associé à une formule SPBN

associe un ensemble de $Nb(p,\phi)$ éléments noté $S(p,\phi)$ à chaque variable $p \in \mathcal{P}(\phi)$, tel que pour chaque $p,q \in \mathcal{P}(\phi)$ avec $p \neq q$, $S(p,\phi) \cap S(q,\phi) = \emptyset$.

Soit ϕ une formule PBN, on note $G_\phi^{\mathcal{O}} = (V,E)$ le graphe non orienté associé à la formule ϕ, avec $V = \bigcup_{p\in\mathcal{P}(\phi)} S(p,\phi)$ et pour chaque $v,v' \in V$, $\{v,v'\} \in E$ si et seulement s'il existe $p,q \in \mathcal{P}(\phi)$ tel que $p \neq q$, $v \in S(p,\phi)$, $v' \in S(q,\phi)$ et p et q appartiennent à la même clause.

Par exemple, le graphe présenté dans la figure 3 correspond à la formule SPBN de l'exemple 1. L'ensemble des sommets V est obtenu en prenant en compte l'union de $S(x,\phi)$ pour chaque $x \in \mathcal{P}(\phi)$. A titre d'illustration, $S(p,\phi) = \{v_p^1, v_p^2, v_p^3\}$, les variables restantes apparaissent uniquement une seule fois dans les clauses positives ($S(q,\phi) = \{v_q\}$). Concernant les arêtes, on a les variables p et q qui appartiennent à la même clause, donc on ajoute des arêtes entre les occurrences de p ($\{v_p^1, v_p^2, v_p^3\}$) et q ($\{v_q\}$).

Proposition 4. *Soit ϕ une formule PBN, $G_\phi^{\mathcal{O}} = (V,E)$ son graphe associé, $p \in \mathcal{P}(\phi)$ et $v \in S(p,\phi)$. Si M est un stable maximum tel que $v \in S$, alors $S(p,\phi) \subseteq M$.*

Preuve. Supposant qu'un sommet $v' \in S(p,\phi)$ existe, tel que $v \neq v'$ et $v' \notin M$. Alors, il existe $v'' \in M$ tel que $\{v',v''\} \in E$ vu que M est un stable maximum. Donc par définition $\{v,v''\} \in E$ est vérifié, contradiction. □

Le théorème 4 est une généralisation du théorème 1. En utilisant la proposition 2, il suffit de satisfaire un seul littéral dans chaque clause positive pour avoir un modèle pour une formule SPBN, en combinant cette propriété avec la proposition 4 on obtient une réduction d'une instance SPBN-SAT vers une instance du problème de recherche d'un stable maximum.

Théorème 4. *Soit ϕ une formule SPBN, ϕ est satisfiable si et seulement si $\alpha(G_\phi^{\mathcal{O}}) \geqslant n$ où $n = |Pos(\phi)|$.*

Preuve. La preuve est similaire à celle du théorème 1, la seule différence réside dans le fait de prendre en compte les occurrences des littéraux positifs vu que les clauses positives partagent au plus un littéral dans les formules SPBN.

Partie ⇒. En utilisant la proposition 2, ϕ est satisfiable si et seulement si $\mathcal{R}(\phi)$ l'est. Soit $\mathcal{I}$ une interprétation booléenne tel que $\mathcal{I}(\mathcal{R}(\phi)) = 1$ et $S = \{v \in S(p,\phi) \mid p \in \mathcal{P}(\phi) \text{ et } \mathcal{I}(p) = 1\}$. Vu que $\mathcal{I}$ satisfait les clauses positives, $|S| \geqslant n$ est vérifié. Rappelons que les clauses positives partagent au plus un littéral. Supposant maintenant qu'il existe deux sommets $v,v' \in S$ tel que $\{v,v'\}$ est une arête dans $G_\phi^{\mathcal{O}}$. Donc, il existe une clause contenant $\neg p \vee \neg q$ dans $\mathcal{R}(\phi)$ telle que $v \in S(p,\phi)$ et $v' \in S(q,\phi)$. Contradiction, car $\mathcal{I}(p) = 0$ ou $\mathcal{I}(q) = 0$.

Partie ⇐. Soit M un stable maximum de $G_\phi^{\mathcal{O}}$ avec $|M| \geqslant n$. On définit une interprétation booléenne $\mathcal{I}$ de ϕ comme suit :

$$\mathcal{I}(p) = \begin{cases} 1 & \text{si } S(p,\phi) \subseteq M \\ 0 & \text{sinon} \end{cases}$$

Puis, en utilisant la définition de $G_\phi^{\mathcal{O}}$, $\mathcal{I}$ satisfait les clauses binaires négatives car, pour chaque $\neg p \vee \neg q \in Nég(\phi)$, $S(p,\phi) \cap M = \emptyset$ ou $S(q,\phi) \cap M = \emptyset$. De plus $M \subseteq \bigcup_{p\in\mathcal{P}(\phi)} S(p,\phi)$ et en utilisant la proposition 4, on a soit $S(p,\phi) \subseteq M$ ou $S(p,\phi) \cap M = \emptyset$ pour chaque $p \in \mathcal{P}(\phi)$. De plus, en utilisant la définition de $G_\phi^{\mathcal{O}}$, pour chaque $c \in Pos(\phi)$ et pour chaque $p,q \in c$ avec $p \neq q$, $S(p,\phi) \cap M = \emptyset$ ou $S(q,\phi) \cap M = \emptyset$. Donc, $\mathcal{I}$ satisfait au plus un littéral dans chaque clause positive. Vu que $|M| \geqslant n$ on déduit que $\mathcal{I}$ satisfait les clauses positives de ϕ. □

Reprenons la formule SPBN ϕ donnée dans l'exemple 1, son graphe associé $\mathcal{G}_\phi^{\mathcal{O}}$ est représenté dans la figure 3. L'ensemble de sommets $S = \{v_p^1, v_p^2, v_p^3, v_t\}$ est un stable maximum de $\mathcal{G}_\phi^{\mathcal{O}}$. En utilisant le théorème 4, ϕ est satisfiable vu qu'elle contient 4 clauses positives et $\alpha(\mathcal{G}_\phi^{\mathcal{O}}) = 4$. Le modèle de ϕ induit par S est $\{p,t,\neg q,\neg r,\neg s,\neg u\}$.

Comme avec le théorème 1, le théorème 4 nous permet de définir des classes traitables pour le problème SPBN-SAT en utilisant des classes traitables dans le problème de recherche d'un stable maximum.

6 Classes traitables pour les formules PBN

Une généralisation de nos précédents résultats sera présentée dans cette section. Pour cela, nous définissons une classe particulière des formules PBN, que

nous appelons Clause-Couverte PBN (CC-PBN), généralisant celle des formules SPBN. Cette généralisation permet aux clauses positives d'une formule PBN de partager plus d'une variable.

Soit ψ un ensemble de clauses positives et p une variable propositionnelle. $\mathcal{C}(p, \psi)$ représente l'ensemble de clauses de ψ où p apparaît, i.e., $\mathcal{C}(p, \psi) = \{c \in \psi \mid p \in c\}$. On dit que $\mathcal{C}(p, \psi)$ est la *couverture* de p dans ψ.

Définition 4 (Formule CC-PBN)**.** *Une* formule CC-PBN ϕ *est une formule PBN où, pour chaque* $p, q \in \mathcal{P}(\phi)$ *avec* $p, q \in c$ *pour une clause* $c \in Pos(\phi)$*, on a* $\mathcal{C}(p, Pos(\phi)) \subseteq \mathcal{C}(q, Pos(\phi))$ *ou bien* $\mathcal{C}(q, Pos(\phi)) \subseteq \mathcal{C}(p, Pos(\phi))$*.*

Autrement dit, Une formule CC-PBN est une formule PBN où pour chaque deux littéraux distincts dans une clause positive, la *couverture* de l'un est incluse dans celle de l'autre dans la partie positive.

Il est clair que les formules SPBN sont des formules CC-PBN. En effet, si ϕ est une formule SPBN alors pour chaque clause $c \in Pos(\phi)$, il existe un littéral p tel que $\{c\}$ est un sous ensemble de $\mathcal{C}(p, Pos(\phi))$, i.e., $\mathcal{C}(p, Pos(\phi)) \supset \{c\}$.

Exemple 2. *La formule* ϕ *suivante est une formule CC-PBN :*
$(p_1 \vee p_2 \vee p_3 \vee p_4) \wedge$
$(p_1 \vee p_2 \vee p_3 \vee p_5) \wedge$
$(p_1 \vee p_2 \vee p_6 \vee p_7) \wedge$
$(p_1 \vee p_8 \vee p_9 \vee p_{10}) \wedge$
$(\neg p_1 \wedge \neg p_4) \wedge$
$(\neg p_3 \vee \neg p_7) \wedge$
$(\neg p_5 \vee \neg p_6) \wedge$
$(\neg p_2 \vee \neg p_{10})$.

Il est clair que $\mathcal{C}(p_3, Pos(\phi)) \subset \mathcal{C}(p_2, Pos(\phi)) \subset \mathcal{C}(p_1, Pos(\phi))$ *et les autres variables apparaissent une seule fois dans* $Pos(\phi)$*.*

Nous montrons qu'il suffit de satisfaire exactement un littéral dans chaque clause positive afin d'obtenir un modèle pour une formule CC-PBN.

Proposition 5. *Soit* ϕ *une formule CC-PBN,* ϕ *est satisfiable si et seulement si* $\mathcal{R}(\phi)$ *est satisfiable.*

Preuve. Il suffit de monter que si ϕ est satisfiable alors $\mathcal{R}(\phi)$ l'est aussi, l'inverse est trivial. Soit $\mathcal{I}$ un modèle de ϕ et $\preceq$ un ordre sur $\mathcal{P}(\phi)$ satisfaisant les propriétés suivantes : pour chaque $p, q \in \mathcal{P}(\phi)$, si $\mathcal{C}(q, Pos(\phi))$ est un sous ensemble de $\mathcal{C}(p, Pos(\phi))$ ($\mathcal{C}(q, Pos(\phi)) \subset \mathcal{C}(p, Pos(\phi))$) alors $p \preceq q$. Pour une clause positive $c \in Pos(\phi)$, $\ell_{c,\mathcal{I}}$ désigne la plus petite variable propositionnelle dans c en respectant $\preceq$ tel que $\mathcal{I}(\ell_{c,\mathcal{I}}) = 1$. Soit $\mathcal{I}'$ une interprétation booléenne de $\mathcal{R}(\phi)$ définie comme suit :

$$\mathcal{I}'(p) = \begin{cases} 1 & \text{si } \exists c \in Pos(\phi), p = \ell_{c,\mathcal{I}} \\ 0 & \text{sinon} \end{cases}$$

Sachant que, pour chaque $p, q \in \mathcal{P}(\phi)$ avec $p, q \in c$ et $c \in Pos(\phi)$, on a $\mathcal{C}(p, Pos(\phi)) \subseteq \mathcal{C}(q, Pos(\phi))$ ou bien $\mathcal{C}(q, Pos(\phi)) \subseteq \mathcal{C}(p, Pos(\phi))$, $\mathcal{I}$ satisfait exactement un littéral dans chaque clause positive. Par conséquent $\mathcal{I}'$ est un modèle de $\mathcal{R}(\phi)$. □

Généralisant le théorème 4, en combinant la proposition 4 et la proposition 5.

Théorème 5. *Soit* ϕ *une formule CC-PBN,* ϕ *est satisfiable si et seulement si* $\alpha(G^{\mathcal{O}}_{\phi}) \geqslant n$ *où* $n = |Pos(\phi)|$*.*

Preuve. La preuve est similaire à celle du théorème 4.
Partie $\Rightarrow$**.** La preuve est obtenue en remplaçant la proposition 2 par la proposition 5 et en utilisant les mêmes arguments.
Partie $\Leftarrow$**.** C'est exactement les mêmes arguments et l'interprétation booléenne suivante obtenue à partir d'un stable M de $G^{\mathcal{O}}_{\phi}$ est un modèle de ϕ, si la taille du stable est supérieure ou égale à n :

$$\mathcal{I}(p) = \begin{cases} 1 & \text{si } S(p, \phi) \subseteq M \\ 0 & \text{sinon} \end{cases}$$

□

Pour tous les fragments de la hiérarchie, IPBN, SPBN et CC-PBN, le théorème 5, nous permet de trouver les contreparties SAT des différentes classes traitables du problème de recherche d'un stable maximum.

7 Connexion avec les modèles minimaux

Le problème du modèle minimal a des applications importantes dans l'IA, comme la circonscription propositionnelle et le diagnostic minimal [29, 34]. Nous présentons ici une caractérisation des modèles minimaux dans le cas des formules CC-PBN en utilisant le problème de recherche d'un stable maximum.

Soient $\mathcal{I}$ et $\mathcal{I}'$ deux interprétations d'une formule propositionnelle ϕ. $\mathcal{I}$ est dit plus petit que $\mathcal{I}'$, noté $\mathcal{I} \preceq \mathcal{I}'$, si $\{p \in \mathcal{P}(\phi) \mid \mathcal{I}(p) = 1\} \subseteq \{p \in \mathcal{P}(\phi) \mid \mathcal{I}'(p) = 1\}$.

Définition 5. *Soit* ϕ *une formule propositionnelle et* $\mathcal{I}$ *un modèle de* ϕ*.* $\mathcal{I}$ *est dit modèle minimal de* ϕ *s'il n'existe pas de modèle strictement plus petit que* $\mathcal{I}$

Proposition 6. *Soit ϕ une formule CC-PBN et $\mathcal{I}$ un modèle de ϕ. $\mathcal{I}$ est un modèle minimal de ϕ si et seulement si $\mathcal{I}$ satisfait exactement un littéral dans chaque clause positive.*

Preuve. **Partie** $\Leftarrow$. Supposons qu'il existe un modéle $\mathcal{I}'$ de ϕ tel que $\{p \in \mathcal{P}(\phi) \mid \mathcal{I}'(p) = 1\} \subset \{p \in \mathcal{P}(\phi) \mid \mathcal{I}(p) = 1\}$. Alors, on obtient une contradiction vu qu'il existe au moins une clause dans $Pos(\phi)$ non satisfaite. En effet, vu que $\mathcal{I}$ satisfait exactement un littéral dans chaque clause positive, les clauses positives qui contiennent les littéraux dans $\{p \in \mathcal{P}(\phi) \mid \mathcal{I}(p) = 1\} \setminus \{p \in \mathcal{P}(\phi) \mid \mathcal{I}'(p) = 1\}$ ne sont pas satisfaites par $\mathcal{I}'$. Nous rappelons que nous considérons que les formules PBN ne contenant aucun littéral pure.
Partie $\Rightarrow$. Supposons qu'il existe une clause positive dans $Pos(\phi)$ contenant deux littéraux p et q tel que $\mathcal{I}(p) = \mathcal{I}(q) = 1$. Vu que ϕ est une formule CC-PBN, on sait que $\mathcal{C}(p, Pos(\phi)) \subseteq \mathcal{C}(q, Pos(\phi))$ ou $\mathcal{C}(q, Pos(\phi)) \subseteq \mathcal{C}(p, Pos(\phi))$. On considère, sans perte de généralité, $\mathcal{C}(p, Pos(\phi)) \subseteq \mathcal{C}(q, Pos(\phi))$. Donc, l'interprétation $\mathcal{I}'$ suivante est un modèle de $\phi : \mathcal{I}'(p) = 0$ et $\mathcal{I}'(r) = \mathcal{I}(r)$ pour chaque variable propositionnelle $r \neq p$. Donc, $\mathcal{I}$ n'est pas un modèle minimal de ϕ, contradiction. □

Théorème 6. *Soit ϕ une formule CC-PBN et $\mathcal{I}$ une interprétation booléenne de ϕ. $\mathcal{I}$ est un modèle minimal de ϕ si et seulement si $\alpha(G^{\mathcal{O}}_{\phi}) = |Pos(\phi)|$ et $\bigcup_{p \in E} S(p, \phi)$ est un stable maximum de $G^{\mathcal{O}}_{\phi}$ où $E = \{p \in \mathcal{P}(\phi) \mid \mathcal{I}(p) = 1\}$.*

Preuve. **Partie** $\Leftarrow$. En utilisant le théorème 5, on sait que ϕ est satisfiable vu que $\alpha(G^{\mathcal{O}}_{\phi}) = |Pos(\phi)|$. De plus $\mathcal{I}$ est un modèle de ϕ (preuve du théorème 5). Avec la proposition 6, $\mathcal{I}$ est un modèle minimal de ϕ vu que $\mathcal{I}$ satisfait exactement un littéral dans chaque clause positive ($\alpha(G^{\mathcal{O}}_{\phi}) = |Pos(\phi)|$).
Partie $\Rightarrow$. En utilisant la proposition 6, $\mathcal{I}$ satisfait exactement un littéral dans chaque clause positive. Donc, $|\bigcup_{p \in E} S(p, \phi)| = |Pos(\phi)|$ et $\bigcup_{p \in E} S(p, \phi)$ est un stable de $G^{\mathcal{O}}_{\phi} = (V, E)$. Avec la définition de $G^{\mathcal{O}}_{\phi}$, on sait que chaque sommet de V est soit dans $\bigcup_{p \in E} S(p, \phi)$ soit adjacent à un sommet de $\bigcup_{p \in E} S(p, \phi)$. Ainsi, ce stable est un stable maximum de $G^{\mathcal{O}}_{\phi}$. □

8 Travaux connexes

Plusieurs travaux ont étudié les relations de résolution, d'inférence et de traitabilité entre le problème SAT et le problème de satisfaction de contraintes (CSP) en utilisant plusieurs encodages (par exemple [38, 32]). La relation entre notre approche et les CSP binaires ne concerne que le fragment le plus spécifique de la hiérarchie, les formules IPBN. En effet, ce fragment particulier concerne les formules PBN où les clauses positives ne partagent pas de variables. Notre approche l'étend, en introduisant des formes imbriquées obtenues en considérant quelques propriétés sur les occurrences de variables dans la partie positive d'une formule PBN. En analysant la partie positive des formules PBN, des résultats sur la traitabilité du problème peuvent être obtenus. C'est le point le plus important de ce papier.

Il est bien connu que la résolution d'une instance CSP binaire P est équivalente à trouver un stable dans le complément de la microstructure de P [33]. Il est également bien connu qu'une instance CSP binaire peut être codée comme une instance IPBN-SAT en utilisant l'encodage direct et inversement [38]. Par conséquent, des classes traitables du problème de recherche d'un stable maximum pour différentes classes de graphes peuvent être utilisées pour caractériser des classes traitables en CSP [8, 13]. Donc, des classes traitables dans IPBN-SAT peuvent être dérivées de la même manière.

Rappelons que la microstructure d'une instance CSP binaire est le graphe (V, E) où V est l'ensemble des affectations possibles de valeurs aux variables et E relie toutes les paires d'affectations possibles. Le complément de la microstructure est le complément de ce graphe. Il est intéressant de noter que la représentation de la microstructure joue un rôle important dans la caractérisation des classes traitables dans les CSP, en imposant des conditions dans la microstructure (par exemple [35, 9]). Plus généralement, plusieurs résultats sont obtenus grâce à des réductions polynomiales entre différents problèmes NP-complet ou NP-difficile (par exemple [24]). Citons par exemple les travaux de Jégou et Paris [23], où une instance SAT est représentée sous la forme d'un graphe non orienté issu d'une transformation polynomiale de SAT en problème de la clique. En exploitant les propriétés bien connues des graphes cordaux, les auteurs caractérisent une nouvelle classe traitable pour SAT. Une classe traitable bien connue, appelée les formules CNF appariées [19], est caractérisée par des notions de la théorie de graphe. Une formule CNF est appariée si son graphe biparti associé (dont les sommets sont des clauses et des variables, et les arêtes relient les variables aux clauses où elles apparaissent) contient un couplage (*matching*) qui couvre toutes les clauses. D'autres réductions polynomiales d'un problème π à un problème π' sont également utilisées pour bénéficier des techniques de résolution de π'. Citons par exemple la réduction du problème de recherche d'un stable maximum dans un graphe au problème de satisfiabilité minimum (MINSAT) proposé

par Ignatiev et al. [22].

9 Conclusion et perspectives

Dans ce papier, nous avons proposé un nouveau cadre pour caractériser de nouvelles classes traitables pour le problème SAT. Sans perte de généralité, nous nous concentrons sur une forme particulière de formules booléennes avec seulement des clauses positives et des clauses binaires négatives. Nous avons défini une hiérarchie de fragments caractérisés par des contraintes spécifiques sur les occurrences des variables positives. Pour chaque fragment et pour chacune de ses formules propositionnelles, on donne une réduction à un graphe, où la satisfiabilité de cette formule est équivalente à l'existence d'un stable maximum de taille supérieure ou égale au nombre de clauses positives. Cette propriété vient du fait que dans la forme des formules considérée nous avons seulement besoin de satisfaire un littéral dans chaque clause positive. Cette forme nous permet de caractériser de nouvelles classes traitables à partir des algorithmes polynomiaux existants du problème de recherche d'un stable maximum dans certaines classes de graphes, comme les graphes sans étoile et les graphes parfaits. De plus, nous proposons une caractérisation des modèles minimaux basées sur le problème de recherche d'un stable maximum.

Notre approche peut être utilisée pour trouver de nouveaux liens entre le problème de la satisfiabilité propositionnelle, la théorie des graphes et les problèmes de satisfaction des contraintes. La détermination d'autres propriétés sur la partie positive des formules PBN pourrait mener à de nouvelles classes traitables.

Références

[1] Krishnamurthy Balachander. Shorts proofs for tricky formulas. *Acta Informatica*, 22 :253–275, 1985.

[2] Egon Balas and Chang Sung Yu. On graphs with polynomially solvable maximum-weight clique problem. *Networks*, 19(2) :247–253, 1989.

[3] Belaid Benhamou and Lakhdar Sais. Theoretical study of symmetries in propositional calculus and applications. In *11th International Conference on Automated Deduction (CADE'1992)*, volume 607 of *Lecture Notes in Computer Science*, pages 281–294. Springer, 1992.

[4] Alex Borgida and David W. Etherington. Hierarchical knowledge bases and efficient disjunctive reasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 33–43, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[5] Endre Boros, Peter L. Hammer, and Xiaorong Sun. Recognition of q-Horn formulae in linear time. *Annals of Mathematics and Artificial Intelligence*, 1 :21–32, 1990.

[6] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes : A Survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[7] Marco Cadoli. Panel on "knowledge compilation and approximation" : Terminology, questions, references. In *Proc. of the Fourth Int. Symp. on Artificial Intelligence and Mathematics, AI/Math-96*, pages 183–186, 1996.

[8] 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, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, pages 807–811, 2003.

[9] David A. Cohen, Martin C. Cooper, Páidí Creed, Dániel Marx, and András Z. Salamon. The tractability of CSP classes defined by forbidden patterns. *J. Artif. Intell. Res. (JAIR)*, 45 :47–78, 2012.

[10] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[11] Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8(4) :28–32, October 1976.

[12] Martin Cooper, Achref El Mouelhi, Cyril Terrioux, and Bruno Zanuttini. On Broken Triangles (regular paper). In *International Conference on Principles and Practice of Constraint Programming (CP'2014)*, pages 9–24, 2014.

[13] Martin C. Cooper and Stanislav Zivny. A new hybrid tractable class of soft constraint problems. In *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, pages 152–166, 2010.

[14] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2) :125–150, 2000.

[15] Peter Damaschke, Haiko Müller, and Dieter Kratsch. Domination in convex and chordal bipartite graphs. *Information Processing Letter*, 36(5) :231–236, December 1990.

[16] Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artif. Intell.*, 56(2-3) :197–222, 1992.

[17] Wiliam F. Dowling and Jean H. Gallier. Linear-time algorithms for testing satisfiability of propositional Horn formulae. *Journal of Logic Programming*, pages 267–284, 1984.

[18] Ralph Faudree, Evelyne Flandrin, and Zdeněk Ryjáček. Claw-free graphs — a survey. *Discrete Mathematics*, 164(1–3) :87 – 147, 1997.

[19] John Franco and Allen Van Gelder. A perspective on certain polynomial-time solvable classes of satisfiability. *Discrete Applied Mathematics*, 125(2–3) :177 – 214, 2003.

[20] Martin Charles Golumbic, editor. *Algorithmic Graph Theory and Perfect Graphs*. Annals of Discrete Mathematics, 2004.

[21] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2) :169–197, 1981.

[22] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. On reducing maximum independent set to minimum satisfiability. In *17th International Conference on Theory and Applications of Satisfiability Testing (SAT'2014)*, pages 103–120, 2014.

[23] Philippe Jégou and Lionel Paris. A new formula rewriting by reasoning on a graphical representation of SAT instances. In *Eighth Symposium on Abstraction, Reformulation, and Approximation, SARA 2009, Lake Arrowhead, California, USA, 8-10 August 2009*, 2009.

[24] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[25] Melven R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2) :15–20, 1967.

[26] Harry R. Lewis. Renaming a set of clauses as a horn set. *J. ACM*, 25(1) :134–135, January 1978.

[27] Vladimir Lifschitz. Computing circumscription. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85)*, pages 121–127, 1985.

[28] Vadim V. Lozin and Martin Milanič. A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. *Journal of Discrete Algorithms*, 6(4) :595 – 604, 2008.

[29] John McCarthy. Circumscription - a form of non-monotonic reasoning. *Artif. Intell.*, 13(1-2) :27–39, 1980.

[30] George J. Minty. On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory, Ser. B*, pages 284–304, 1980.

[31] Sebastian Ordyniak, Daniel Paulusma, and Stefan Szeider. Satisfiability of acyclic and almost acyclic {CNF} formulas. *Theoretical Computer Science*, 481(0) :85 – 99, 2013.

[32] Justyna Petke and Peter Jeavons. The order encoding : From tractable CSP to tractable SAT. In *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, pages 371–372, 2011.

[33] Jégou Philippe. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, USA, July 11-15, 1993.*, pages 731–736, 1993.

[34] Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1) :57–95, 1987.

[35] András Z. Salamon and Peter G. Jeavons. Perfect constraints are tractable. In *Principles and Practice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia, September 14-18, 2008. Proceedings*, pages 524–528, 2008.

[36] Bart Selman and Henry Kautz. Knowledge compilation using horn approximations. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 904–909. MIT Press, 1991.

[37] Juraj Stacho. 3-colouring at-free graphs in polynomial time. In *Algorithms and Computation*, volume 6507 of *Lecture Notes in Computer Science*, pages 144–155. Springer Berlin Heidelberg, 2010.

[38] Walsh Toby. SAT v. CSP. In *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings*, pages 441–456, 2000.

Modernisation de la duplication de clauses

Guillaume Baud-Berthier[12] **Laurent Simon**[1]

[1] Université de Bordeaux, LaBRI
[2] SafeRiver
guillaume.baud-berthier@safe-river.com lsimon@labri.fr

Résumé

Le Model Checking est une méthode de vérification formelle très populaire dans le milieu industriel, en raison de son automatisation et de sa faculté à produire une trace d'exécution courte lorsqu'il existe un comportement non désiré. Le model checking borné (BMC) et la k-induction sont deux méthodes paramétrées, allant de pair. Elles sont toutes les deux très dépendantes de l'efficacité de leur procédure de décision sous-jacente, généralement un solveur SAT/SMT, en raison notamment de leur propension à générer des formules de plus en plus grandes. Ces formules ont une caractéristique particulière : elles peuvent être trivialement partitionnées en 3 ensembles, dont l'un possède une structure symétrique. Dans cet article, nous proposons d'exploiter cette structure symétrique en dupliquant les clauses apprises par le solveur. Cette idée fut déjà proposée peu après l'introduction de BMC, mais rapidement abandonnée à cause de sa tendance à surcharger le solveur SAT avec un nombre trop important de clauses. Nous proposons d'étendre la duplication pour la k-induction et suggérons une méthode simplifiée, i.e. sans incidence sur le solveur, pour détecter les clauses duplicables. En outre, nous montrons qu'en sélectionnant soigneusement les clauses à dupliquer et à quels temps les dupliquer, nous obtenons des gains intéressants dans notre model checker, ce qui va à l'encontre de l'idée communément admise.

Abstract

Model Checking is a well-established formal verification method in industry, mainly because of its automation and its ability to produce short execution trace when a bug is found. Bounded Model Checking (BMC) and k-induction are both parameterized methods, working together, and relying almost entirely on their underlying decision procedure, e.g. SAT/SMT solver. The formulas generated by those methods can be trivially partitioned into 3 sets, and such that one of these set is perfectly symmetric. In this paper, we propose to exploit this symmetry to replicate learnt clauses. This was already suggested in the early years of BMC, but quickly left aside because of its tendency to overburden the solver with too many clauses. In this paper, we extend replication to k-induction and propose a simpler but sound strategy to detect replicable clauses. Moreover, we show that by carefully selecting learnt clauses to replicate and where to replicate them, we can observe an interesting progress in our model checker.

1 Introduction

Le model checking est une technique de vérification formelle très utilisée dans le milieu industriel, pour vérifier les circuits électroniques, ou plus généralement les systèmes informatiques dits *critiques*. Ceci est principalement dû à son caractère automatique et à sa capacité de générer de courtes traces d'exécution menant à un comportement non désiré (*bug*). Le model checking borné (BMC) [4] est une méthode paramétrée qui vérifie exhaustivement, dans un système, l'absence de chemins d'une taille donnée menant à un *mauvais* état. Il s'agit d'une des premières applications industrielles et certainement du premier succès historique des solveurs SAT (avec la planification). Cependant, BMC est un algorithme orienté vers la recherche de *contre-exemple* et n'est généralement pas utilisé pour en prouver l'absence [1], ce qui a conduit à l'introduction de la *k-induction* [12], qui est basée sur le principe de récurrence, et ajoute donc un critère de terminaison : l'idée est de montrer que, lorsque le système vérifie le comportement souhaité pendant un nombre de cycle donné (cette hypothèse est fournie par BMC), alors on ne peut plus atteindre un mauvais état. Les performances de ces deux méthodes de model checking sont très dépendantes des performances de la procédure de décision sous-jacente, ici le solveur SAT.

1. En effet, il est très difficile de déterminer la profondeur séquentielle d'un système.

Ces méthodes, générant des formules SAT extraordinairement grandes (pouvant dépasser plusieurs milliards de clauses et des millions de variables), ou demandant des milliers d'appels SAT sur une même instance, ont poussé un changement de paradigme dans la conception des solveurs SAT. Les solveurs SAT "modernes", ou Conflict-Driven Clause Learning (CDCL) [11] ont été initialement conçus pour résoudre de telles instances.

Ce papier s'inscrit dans une lignée de travaux visant à interconnecter le model checker et le solveur SAT. Plus précisement, l'objectif est d'améliorer les performances du solveur en utilisant des éléments d'informations de plus haut niveaux, i.e. disponible au niveau du model checker. Aux prémices de BMC, plusieurs idées ont émergé [13, 14, 16] :

1. Réutilisation des clauses apprises lors des appels successifs BMC avec une profondeur de plus en plus grande : *SAT incrémental*.
2. Remplacement de l'heuristique de sélection des variables dans le solveur par un ordre statique basé sur le modèle, e.g. brancher en priorité sur les entrées.
3. Exploiter la structure symétrique de la formule BMC pour dupliquer les clauses apprises par le solveur.

La résolution incrémentale est, de nos jours, utilisée dans presque tous les model checkers effectuant BMC et la k-induction. La sélection des variables fut abandonnée en faveur de VSIDS [11] ayant une plus grande adaptabilité, fournissant ainsi de meilleurs performances de manière générale. Dans cet article, nous nous intéressons plus particulièrement à la duplication de clauses. Cette dernière fut abandonnée en raison de sa propension à générer beaucoup de clauses dupliquées, ce qui entraine une surcharge du solveur et par conséquent ralentit la propagation. Notre regain d'intérêt pour cette technique provient de deux nouvelles caractéristiques des solveurs SAT récents : la suppression aggressive de la base de données des clauses apprises et le score *LBD (Literal Block Distance)* permettant d'évaluer la *qualité* des clauses [1]. Les contributions de ce papier sont les suivantes :

— Extension de la duplication des clauses à l'algorithme ZigZag (BMC et k-induction), tout en considérant les contraintes de chemin simple.
— Simplification de la méthode de caractérisation des clauses duplicables, en d'autres mots, comment déterminer si une clause est duplicable ou non.
— Sélection des clauses à dupliquer en fonction de leur taille et de leur score LBD, ainsi que le choix de la base de données dans laquelle ajouter ces clauses dupliquées.

Le reste de l'article est organisé comme suit. Dans un premier temps, nous rappelons les principes généraux des solveurs SAT et des algorithmes de vérification de modèle. Dans la partie 3, nous présentons de façon informelle la duplication et nous faisons un état de l'art des critères de duplicabilité. Dans la partie 4, nous proposons une nouvelle approche pour détecter les clauses duplicable et appliquons cette méthode à l'algorithme ZigZag. Dans la partie 5, nous vérifions expérimentalement l'efficacité de cette approche. Enfin, nous concluons dans la dernière partie.

2 Définitions

Le problème de satisfaisabilité (SAT) est un problème NP-complet qui vise à déterminer s'il existe une affectation de valeurs aux variables d'une formule propositionnelle rendant cette formule vraie. Malgré sa complexité théorique intrinsèque, les progrès récents dans sa résolution en pratique en ont fait une des méthodes les plus utilisées pour résoudre les problèmes difficiles, notamment dans le milieu de la vérification formelle.

Une formule propositionnelle est constituée d'un ensemble de variables booléennes, des constantes logiques : $\top$ (vrai, 1) et $\bot$ (faux, 0), et des opérateurs logiques : $\neg$ (non), $\vee$ (ou), $\wedge$ (et). Un *littéral* est une variable ou sa négation : x, $\neg x$. Une clause est une disjonction de littéraux, e.g. $x \vee \neg y \vee z$. Une formule est dite sous *forme normal conjonctive* (CNF) si elle est une conjonction de clause, e.g. $(x \vee \neg y \vee z) \wedge (\neg x) \wedge (y \vee z)$. Nous pouvons également considérer une clause comme un ensemble de littéraux, e.g. $\{x, \neg y, z\}$, et une CNF comme un ensemble de clauses : $\{\{x, \neg y, z\}, \{\neg x\}, \{y, z\}\}$. Une affectation des variables est une fonction associant à chaque variable une valeur booléenne. Une affectation satisfait une formule si la formule s'évalue à $\top$ en substituant chaque variable par sa valeur associée.

2.1 Principes généraux des solveurs SAT

Un solveur SAT est souvent utilisé comme une boîte noire très optimisée, résolvant le problème SAT. Cette utilisation en "boîte noire" est d'ailleurs l'une des raisons du succès de SAT. La plupart des solveurs prennent ainsi en entrée une formule sous forme CNF et sont capables de produire une affectation des variables si la formule est satisfaisable, sans aucune connaissance sur le sens des variables et des clauses encodées (en général : certaines approches specialisées ne rentrent pas dans ce cadre). Les solveurs plus récents peuvent également produire une preuve de non-satisfaisabilité. Cependant, pour être capable par la

suite de caractériser si une clause est duplicable ou non, nous avons besoin d'introduire quelques notions essentielles des solveurs modernes.

Un des premiers solveurs SAT, DP [7], éliminait les variables une par une, à l'aide de la *règle de résolution*, jusqu'à trouver la clause vide (UNSAT), ou jusqu'à ce que la formule devienne vide (SAT). Cependant, en raison du problème de l'explosion combinatoire en mémoire, cette technique fut rapidement abandonnée en faveur de la recherche avec saut arrière (DPLL [6]). Notons tout de même que l'élimination de variables reste une étape de pré-traitement quasi-indispensable dans tous les solveurs modernes. L'élimination d'une variable consiste simplement à effectuer toutes les résolutions possibles sur cette variable, puis à retirer de la formule toutes les clauses contenant cette variable, et ajouter toutes les clauses des résolutions. En d'autres termes, l'élimination de la variable x dans la formule Σ se résume à appliquer ce pseudo-algorithme : D'abord, partitionner Σ en 3 ensembles $\Sigma_{\setminus x}$ l'ensemble des clauses ne contenant pas la variable x, Σ_x l'ensemble des clauses contenant le littéral x et $\Sigma_{\neg x}$ l'ensemble des clauses contenant le littéral $\neg x$. Puis, supprimer toutes les occurrences de x et $\neg x$ dans Σ_x et $\Sigma_{\neg x}$, et enfin reconstruire $\Sigma = \Sigma_{\setminus x} \wedge (\Sigma_x \vee \Sigma_{\neg x})$. Pendant l'étape de pré-traitement, l'élimination de variables est usuellement appliquée sur toutes les variables pour lesquelles le nombre de clauses de la formule n'augmente pas lorsque celles-ci sont éliminées. Ce processus préserve la satisfaisabilité de la formule originale.

Avec l'introduction des algorithmes modernes de résolution du problème SAT, dit CDCL [8, 11, 15], l'apprentissage de clauses devient l'élément clef des solveurs. Cependant, un solveur CDCL peut apprendre plus de 5000 clauses par seconde, c'est pourquoi, il est rapidement devenu crucial de mettre en place des stratégies de gestion pour les bases de données des clauses apprises. Dans *Minisat* [8], les auteurs ont proposés de supprimer les clauses qui étaient les moins vues lors des dernières analyses de conflit. Avec l'idée de mesurer la distance entre les blocs de littéraux (LBD), *Glucose* [2] a introduit une nouvelle façon de classer et donc de supprimer les clauses.

Dans les solveurs modernes, lorsqu'une affectation partielle des variables (générée à l'aide de décisions et de propagations) produit un conflit, i.e. une contradiction sur l'affection d'une variable, une analyse de ce conflit est effectuée. A partir de cette analyse, une clause est apprise pour éviter de regénérer la même affectation partielle. Intuitivement, chaque clause apprise est obtenue par des étapes de résolutions sur un sous-ensemble de clauses propagées au dernier niveau de décision. Des étapes additionnelles de résolutions, utilisées pour réduire la taille de la clause apprise, peuvent également être effectuées (voir [5] pour une référence complète). Le système de score LBD est relativement simple. Le score d'une clause est le nombre de niveaux de décisions distincts de chacun des littéraux de la clause. Dans *Glucose*, les clauses de LBD 2 sont appelées *Glues Clauses*, et ne sont jamais supprimées de la base de données des clauses apprises. Le reste des clauses apprises sont régulièrement supprimées de la base de données, en fonction d'un classement basé sur leur score LBD. Afin de donner un ordre de grandeur, lors d'une exécution de *Glucose*, 95% des clauses apprises peuvent être supprimées au cours de la résolution.

2.2 Model Checking

Le Model Checking est une technique automatique et exhaustive de vérification formelle, qui détermine si un *modèle* vérifie une *propriété* donnée. Les propriétés expriment un comportement particulier du système que l'on souhaite vérifier, et sont généralement définies en logique temporelle. Dans cet article, nous considérons uniquement les modèles représentant des systèmes de transitions booléens à états finis, généralement utilisés pour représenter les circuits séquentielles. En outre, nous nous limitons aux propriétés sous la forme d'assertion, e.g. de la forme AGp en CTL*.

Un système de transitions à états finis est un 4-uplet $\mathcal{M} = \langle V, I, T, P \rangle$, où V est un ensemble de variables booléennes, $I(V)$ et $P(V)$ sont des formules sur V représentant, respectivement, l'ensemble des états initiaux et l'ensemble des états vérifiant la propriété. $T(V, V')$ est la relation de transition, i.e. une formule sur V, V' caractérisant les transitions acceptées du système. $V' = \{v' \mid v \in V\}$ est la version prime de l'ensemble V, généralement utilisée pour représenter les variables au cycle suivant. De la même façon, lorsque nous déplions la relation de transition plusieurs fois, nous utilisons $V_i = \{v_i \mid v \in V\}$ pour définir les mêmes ensembles que V avec renommage des variables. Un état du système est une affectation des variables de V. Les affectations satisfaisant une formule sur V représente donc un ensemble d'états du système. Nous utilisons $Bad(V) = \neg P(V)$ pour exprimer l'ensemble des *mauvais* états du systèmes, i.e. qui contredisent le comportement désiré (la propriété). Un système admet un *contre-exemple* s'il est possible d'atteindre un mauvais état, en partant des états initiaux et en appliquant répétitivement la relation de transition. A l'inverse, on dit que la propriété est vérifiée ou prouvée, s'il n'existe pas de chemin partant des états initiaux et terminant dans un mauvais état. Autrement dit, l'ensemble des états atteignables du système est disjoint de l'ensemble des mauvais états.

2.2.1 BMC

Le model checking borné est un algorithme paramétré orienté vers la recherche de contre-exemples [4]. Intuitivement, l'idée est de construire une formule qui est satisfaisable si le système peut atteindre un mauvais état en k transitions. BMC est très efficace pour trouver des contre-exemples, mais il est difficile de trouver un k suffisamment grand pour garantir que la propriété est vérifiée pour tous les états atteignables du système, tout en étant suffisamment petit pour être résolu en pratique [2]. La définition formelle de BMC est la suivante :

$$\text{BMC}_k = I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge Bad(V_k)$$

Cette formule est ensuite encodée en CNF et sa satisfaisabilité est déterminée à l'aide d'un solveur SAT. Cependant, cette définition ne permet pas de s'assurer qu'il n'existe pas de contre-exemple de profondeur inférieur à k. C'est pourquoi, BMC est généralement exécuté de façon incrémental, i.e. en résolvant BMC_k pour $k = 0, 1, \ldots \infty$, jusqu'à ce qu'un contre-exemple soit trouvé ou que les ressources disponibles soient epuisées. Une autre approche, moins utilisée, consiste à étendre $Bad(V_k)$ à $\bigvee_{i=0}^{k} Bad(V_i)$.

2.2.2 k-induction

Le principe d'induction pour un système de transitions est relativement simple, il s'effectue en 2 étapes :

1. S'assurer que les états initiaux vérifient la propriété (cas de base), formellement :

   $$I(V) \Rightarrow P(V)$$

   ou en *formulation SAT* :

   $$I(V) \wedge \neg P(V) \models \bot$$

2. Puis, s'assurer qu'en appliquant la relation de transition sur l'ensemble des états vérifiant la propriété, alors les états suivants vérifient également la propriété (hérédité), formellement :

   $$P(V) \wedge T(V, V') \Rightarrow P(V')$$

   ou en *formulation SAT* :

   $$P(V) \wedge T(V, V') \wedge \neg P(V') \models \bot$$

Intuitivement, l'induction, qui est équivalente à la 1-induction, montre que si la propriété est vraie pour un cycle donné alors il n'est pas possible d'atteindre un mauvais état en une transition. En d'autres termes, la propriété P est un ensemble dit *inductif* dans le système de transition, i.e. pour tout état vérifiant P, l'ensemble des états atteignables en une transition vérifie également P. Ainsi, par récurrence, P est vrai pour l'ensemble du système.

Cependant, les propriétés sont rarement inductives, ce qui signifie que la formule d'hérédité est satisfaisable. Aucune conclusion ne peut donc être tirée. L'hypothèse de récurrence doit alors être *renforcée*. La k-induction, introduite dans [12], propose de la renforcer en augmentant le nombre de cycles consécutifs pour lesquels P est vrai. BMC est alors utilisé comme *base* du raisonnement, assurant que la propriété est vraie pour les k premiers cycles du système. Puis, l'hérédité en k-induction consiste à vérifier que si la propriété est vraie pendant k cycles consécutifs alors elle reste vraie au cycle suivant. Ainsi, la propriété est prouvée pour l'ensemble du système. Formellement, l'hérédité est définie telle que :

$$k\text{-ind} = \bigwedge_{i=0}^{k-1} \big[P(V_i) \wedge T(V_i, V_{i+1})\big] \wedge Bad(V_k)$$

2.2.3 Algorithme ZigZag

L'algorithme *ZigZag* introduit dans [9] consiste simplement à combiner BMC et la k-induction dans un seul solveur SAT. En effet, les formules générées par BMC et la k-induction possèdent une grande partie commune, et le solveur peut ainsi profiter, en partie, des clauses apprises lors des appels précédents. L'algorithme se résume donc à résoudre consécutivement k-induction et BMC_k, avec $k = 0, 1, \ldots \infty$, jusqu'à ce qu'une formule BMC devienne satisfaisable — il existe alors un contre-exemple — ou qu'une formule k-ind soit insatisfaisable, dans ce cas la propriété est vérifiée pour le système.
Afin de bien illustrer le déroulement de l'algorithme, voici, dans l'ordre, les premiers appels consécutifs du solveur. Par souci de clarté, nous utilisons I_i, P_i pour représenter $I(V_i)$, $P(V_i)$ et $T_{i,j}$ pour $T(V_i, V_j)$.

$$\begin{aligned}
\text{0-ind} &: \quad \big[Bad_0\big] \\
\text{BMC}_0 &: \quad \big[I_0\big] \wedge \big[Bad_0\big] \\
\text{1-ind} &: \quad P_0 \wedge T_{0,1} \wedge \big[Bad_1\big] \\
\text{BMC}_1 &: \quad \big[I_0\big] \wedge P_0 \wedge T_{0,1} \wedge \big[Bad_1\big] \\
\text{2-ind} &: \quad P_0 \wedge T_{0,1} \wedge P_1 \wedge T_{1,2} \wedge \big[Bad_2\big] \\
\text{BMC}_2 &: \quad \big[I_0\big] \wedge P_0 \wedge T_{0,1} \wedge P_1 \wedge T_{1,2} \wedge \big[Bad_2\big] \\
&\vdots
\end{aligned}$$

2. Le nombre d'états du système est évidemment suffisant mais généralement impossible à résoudre en pratique.

Les crochets sont utilisés pour mettre en avant les parties de la formule qui sont temporaires. Ces parties sont généralement *activées/désactivées* à l'aide de littéraux d'activation. Un littéral d'activation est un littéral que l'on ajoute à une ou plusieurs clauses afin de pouvoir utiliser ce littéral pour modifier la prise encompte de ces clauses (en permettant de les satisfaire trivialement). Supposons, par exemple, $C = \{\neg act, x, y\}$ une clause ajoutée à un solveur S, en forçant alors l'affectation de act à $\top$ dans S, on oblige le solveur à prendre en compte la contrainte $x \vee y$ et à l'inverse en forçant act à $\bot$, le solveur peut assigner librement $x = \bot, y = \bot$ en même temps.

2.2.4 Contraintes de chemin simple

Les contraintes de chemin simple, aussi appelées *contraintes sans boucle*, sont ajoutées à la formule de k-induction afin de rendre l'algorithme complet. Elles sont utilisées pour spécifier que chaque état du système dans le chemin construit pour la k-induction doit être différent [3] (Voir [12] pour plus de détails.).

En général, les contraintes *tous différents* génèrent beaucoup de clauses, même lorsque l'introduction de variables est permise. Les auteurs de [9] ont alors proposé de générer ces contraintes *à la demande*. Autrement dit, lorsque le solveur nous fournit une affectation des variables satisfaisant une formule de k-induction, celle-ci est examinée. Ainsi, s'il existe deux états identiques dans l'affectation fournie, une contrainte est ajoutée et le solveur relancé.

Cette partie décrit succinctement la fonction des contraintes de chemin simple. Ces contraintes sont ajoutées dans presque tous les model checkers effectuant la k-induction, il est donc primordial de les considérer dans notre approche. Néanmoins, le seul point important dans le contexte de ce papier est l'introduction de nouvelles variables pour ces contraintes, qui peuvent interférer lors de la duplication.

3 Duplication

Nous présentons d'abord, intuitivement, le fonctionnement de la duplication dans BMC, puis nous donnons quelques détails sur les approches utilisées.

3.1 Structure symétrique et duplication

La duplication de clauses est basée sur la structure symétrique des formules. Les formules générées par BMC contiennent une partie symétrique :

$$I_0 \wedge \underbrace{T_{0,1} \wedge T_{1,2} \wedge \cdots \wedge T_{k-1,k}}_{Partie\ symétrique} \wedge \neg Bad_k$$

En d'autres mots, si on ignore les parties concernant les états initiaux et les mauvais états, la formule est parfaitement symétrique. En effet, il s'agit simplement de la répétition (k fois), ce que l'on appelle parfois le *dépliage*, de la relation de transition. Cela signifie que si le solveur CDCL déduit une nouvelle clause en ne faisant des résolutions que sur la partie symétrique, alors cette nouvelle clause peut être dupliquée à d'autres cycles en renommant simplement les variables. Ces clauses dupliquées empêcheront alors peut-être la même affectation partielle de se produire dans un autre cycle, et accèlerera ainsi potentiellement la résolution du problème.
La notation $C_{[i,j]}$ spécifie que la clause C a été déduite à l'aide de résolution ne faisant intervenir que des clauses provenant des relations de transition et tel que le plus petit (respectivement grand) indice de cycle de l'ensemble des variables utilisées est i (respectivement j). Supposons par exemple, que lors de la résolution de BMC_3, le solveur apprenne la clause $C_{[0,2]}$ à partir de clauses provenant uniquement des relations de transition $T_{0,1}$ et $T_{1,2}$, alors celle-ci peut être dupliquée pour $T_{1,2}$ et $T_{2,3}$ en renommant ses variables :

$$I_0 \wedge \underbrace{T_{0,1} \wedge \overbrace{T_{1,2}}^{C_{[1,3]}}}_{C_{[0,2]}} \wedge T_{2,3} \wedge \neg Bad_3$$

avec $C_{[0,2]} = \{x_0, \neg y_1, x_2\}$ et $C_{[1,3]} = \{x_1, \neg y_2, x_3\}$. La première étape de la duplication se produit donc directement à l'intérieur du processus d'apprentissage du solveur SAT. Ainsi, le solveur est maintenant capable d'apprendre plusieurs clauses par conflit.

En outre, comme BMC est effectué dans un contexte incrémental, une nouvelle relation de transition sera ajoutée au solveur après chaque appel résolu. En sauvegardant les clauses duplicables, ces clauses peuvent alors être dupliquées pour cette nouvelle relation de transition avant même de déterminer la satisfaisabilité de la formule BMC suivante. En reprenant notre exemple précédent, la clause $C_{[2,4]}$ peut être ajoutée, avant d'établir la satisfaisabilité de BMC_4 :

$$I_0 \wedge \underbrace{T_{0,1} \wedge \overbrace{T_{1,2}}^{C_{[1,3]}}}_{C_{[0,2]}} \wedge \underbrace{T_{2,3}}_{C_{[2,4]}} \wedge T_{3,4} \wedge Bad_4$$

Cette seconde étape requiert un plus haut niveau de supervision, extérieur au solveur SAT, et s'effectue directement dans l'algorithme de model checking. La

3. Seules les variables représentant les mémoires sont considérées.

principale difficulté réside donc dans la caractérisation des clauses apprises, à savoir est-ce que cette clause est duplicable et à quels cycles peut-elle être dupliquée.

3.2 Conditions de duplication

Dans les premières tentatives de duplication, une approche consistait à *inciter* l'apprentissage des mêmes clauses à des cycles différents en forçant l'affectation des variables pour mener à des conflits ayant comme conséquence l'apprentissage des clauses souhaitées. Une autre méthode consistait à dupliquer les clauses même si c'est celles-ci n'étaient pas duplicables et de vérifier lors d'un résultat insatisfaisable si les clauses dupliquées n'avaient pas changé la satisfaisabilité de l'instance. A noter que la duplication de clauses non duplicables ne peut entraîner que des faux négatifs [4]. Cependant, ces deux méthodes sont beaucoup trop coûteuses en terme de performance, et donc même si un gain est apporté par les clauses dupliquées, il est annhilé par ce coût.

Par la suite, l'approche qui fut retenue consiste à *marquer* les clauses avec 3 informations additionnelles :

1. un marqueur pour déterminer si la clause est duplicable, i.e. ne dépend pas des états initiaux ou des mauvais états, et n'a pas été déduite à partir de clauses non duplicables.
2. deux marqueurs pour définir le plus petit et le plus grand cycle dont la clause dépend.

Ainsi, lors d'un conflit, si toutes les clauses utilisées lors de l'analyse du conflit sont marquées comme étant duplicables, alors la nouvelle clause l'est aussi. En outre, le plus petit (respectivement grand) cycle de la nouvelle clause correspond au minimum (respectivement maximum) des cycles des clauses impliquées dans le conflit.

La démonstration de la validité de la duplication consiste simplement à montrer que toutes les clauses permettant d'inférer la clause dupliquée se trouvent également dans la formule. En d'autres termes, toutes les clauses utilisées dans l'analyse de conflit pour apprendre la clause que l'on souhaite dupliquer, sont également présentes (modulo renommage) aux cycles où la clause est dupliquée. L'ensemble de ces travaux sont décrits dans [13, 14, 16].

4 Duplication pour l'algorithme ZigZag

La duplication de clauses en model checking n'a pour le moment été expérimentée que pour BMC. Or, la structure de la formule de k-induction s'y prête tout autant, voir plus, car la formule ne contient pas la contrainte des états initiaux. Cependant, comme la k-induction est toujours associée à BMC, nous proposons d'étendre la duplication à l'algorithme ZigZag (partie 2.2.3).

La structure de la formule de l'algorithme ZigZag est la même que celle de BMC :

$$\left[\neg act_{Init} \vee I_0\right] \wedge \bigwedge_{i=0}^{k-1} \left[P_i \wedge T_{i,i+1}\right] \wedge \left[\neg act_{Bad} \vee Bad_k\right]$$

Les conditions de duplication sont donc applicables en suivant le même procédé. Cependant, lors de nos premières expérimentations, nous avons observé que la méthode de marquage des clauses avait un effet préjudiciable sur les performances du solveur. Les opérations nécessaires pour calculer les marqueurs sont pourtant élémentaires. Néanmoins, à chaque analyse de conflit les marqueurs de toutes les clauses qui ont mené aux conflits doivent être comparés. Or, comme ces conflits peuvent se produire plus d'un millier de fois par seconde, les effets en deviennent non négligeables.

Nous proposons une nouvelle approche qui utilise les littéraux d'activation comme marqueur de *duplicabilité*. Nous supprimons également la notion de marqueurs de cycles. De cette façon, nous n'avons plus à gérer le moindre marqueur. Par conséquent, nous obtenons une méthode très simple à mettre en place et les performances du solveur ne sont pas impactées.

L'idée consiste à profiter des littéraux d'activation qui sont ajoutés aux clauses des états initiaux et des mauvais états. Lors de l'analyse d'un conflit, le processus d'apprentissage du solveur va obligatoirement ajouter un des littéraux d'activation à la clause apprise si celle-ci est déduite à partir des états initiaux ou des mauvais états. En effet, sans l'affectation des littéraux d'activation, aucun conflit impliquant les états initiaux et les mauvais états ne peut se produire. Ainsi, lorsque le solveur apprend une clause dépendante de ces états non duplicables, elle contiendra forcément au moins un des littéraux d'activation. Le critère de duplicabilité se réduit donc, dans un premier temps, à la présence ou non d'un littéral d'activation dans une clause apprise. En outre, afin de nous libérer des marqueurs de cycle, nous choisissons de nous restreindre à la duplication incrémentale. Cela revient alors à considérer toutes clauses duplicables comme ayant un marqueur minimum l'indice de cycle 0 et comme marqueur maximum l'indice de cycle courant. La duplication s'effectue alors uniquement au niveau du model checker.

Pour résumer, nous proposons donc une approche ne nécessitant qu'une modification infime du solveur. Pour chaque clause apprise, il suffit de vérifier si elle contient un littéral d'activation, si ce n'est pas le cas,

4. En effet, l'ajout de contraintes ne peut pas rendre la formule satisfaisable si elle ne l'était pas.

```
Fonction ZigZag
    ajouterClauses(I_0 ∨ ¬act_I)
    for i ∈ 0...∞ do
        ajouterClauses(¬P_i ∨ ¬act_Bad_i)
        if solve(act_Bad_i) == UNSAT then
            return 0                          ▷ P est vérifié
        if solve(act_I, act_Bad_i) == SAT then
            return 1                          ▷ Contre-exemple
        ajouterClauses(P_i)
        ajouterClauses(T_{i,i+1})
        dupliquerClauses(i + 1)

Procédure dupliquerClauses(k)
    for all c_l ∈ clauses duplicables do
        ▷ l étant le cycle au cours duquel c est apprise
        c_k ← shiftVariables(c, k − l)
        ajouterClauses(c_k)
```

Algorithme 1: ZigZag avec duplication

elle est duplicable. On sauvegarde alors cette clause en lui associant l'indice du cycle courant auquel elle est apprise. Au niveau du model checker, il est nécessaire d'être en mesure de retrouver pour chaque variable : son cycle et la même variable à un cycle différent. Enfin, entre chaque appel successif au solveur, l'ensemble des clauses sauvegardées est dupliqué comme pour l'exemple suivant : Soit $c_l = \{x_i, y_j\}$ une clause duplicable, l le cycle auquel c a été apprise, et i, j les cycles des variables x, y, et k le cycle auquel on souhaite dupliquer c, on a $i, j \leq l < k$. La version dupliquée de c s'obtient simplement en ajoutant la différence $k - l$ aux indices de cycle des variables : $c_k = \{x_{i+k-l}, y_{j+k-l}\}$. L'algorithme 1 illustre le déroulement de l'approche proposée au niveau du model checker.

La restriction de duplication au niveau du model checker permet de s'émanciper des marqueurs de cycle et fournit également un avantage supplémentaire : elle permet de limiter le nombre de clauses dupliquées, ce qui est le principal inconvénient de la duplication et donc notre objectif premier.

Contraintes de chemin simple à la demande

Comme évoqué précedemment, il est primordial de considérer ces contraintes, si l'on souhaite que la duplication soit adoptée. Cette technique permet l'ajout de nouvelles variables de façon imprévisible, par le fait qu'elles soient générées à la demande. Ces nouvelles variables n'ont donc pas nécessairement d'équivalent dans les autres cycles. La solution que nous avons adoptée pour préserver la satisfaisabilité de la formule consiste à ne pas dupliquer les clauses contenant une variable introduite par ces contraintes. Pour résumer, une clause est considérée comme duplicable si elle ne contient ni une variable introduite par les contraintes de chemin simple, ni un littéral d'activation.

5 Résultats Expérimentaux

Afin de valider expérimentalement notre approche, nous avons implémenté un model checker prenant en entrée des modèles au format standard AIGER [3], et constitué des composants suivants :

- Pré-traitement des modèles AIG fournissant les fonctionnalités habituelles, i.e. propagation de constantes, règles de réécritures (idempotence, contradiction, subsumption, etc.), équivalence structurelle, cône d'influence.
- Elimination des variables à haut-niveau, ce qui permet de désactiver l'élimination des variables au niveau du solveur, afin de ne pas ajouter une clause dupliquée contenant une variable éliminée.
- Algorithme ZigZag (BMC/k-induction)
- Contraintes de chemin simple à la demande
- Duplication des clauses apprises suivant le principe décrit dans la partie 4
- Utilisation du solveur SAT *Glucose* [2] comme procédure de décision

Notre hypothèse est la suivante : la suppression aggressive de clauses et la sélection des clauses à dupliquer permettent de limiter la perte de performance dûe à un trop grand nombre de clauses dupliquées. Afin de la vérifier, nous avons défini un ensemble de stratégies : Ref, O, L, OG5, LG5, O5, L5, tel que :

- Ref correspond à la référence, i.e. sans duplication.
- O et L signifient que les clauses dupliquées sont ajoutées, respectivement, dans la base de données des clauses originales et dans la base des clauses apprises.
- 5 exprime que seules les clauses de tailles inférieures ou égales à 5 sont dupliquées.
- G5 indique qu'en plus des clauses de tailles inférieures ou égales à 5, les glues clauses sont également dupliquées.

A noter que lorsque les clauses sont dupliquées dans la base de données des clauses apprises, nous donnons aux clauses dupliquées le même score LBD que la clause dont elles sont issues, ainsi les glues clauses dupliquées ne sont jamais supprimées.
Nous avons ensuite exécuté toutes ces stratégies sur un ensemble de 513 problèmes (*benchmarks*), provenant de la dernière compétition de model checking hardware (HWMCC 2015), en fixant le temps d'exécution maximal à une heure et la mémoire maximale autori-

	Ref	O	L	OG5	LG5	O5	L5
IND	305	312	308	**300**	**300**	302	302
CEX	108	102	104	**112**	**112**	**112**	110
PRO	92	91	**93**	**93**	**93**	91	**93**
PRF	29	28	31	**34**	31	33	31
TPS	170	255	124	**108**	125	118	114

TABLE 1 – Résultats des différentes stratégies de duplication des clauses apprises sur l'ensemble des instances HWMCC15.

sée à 8Go.

L'ensemble des résultats résumés se trouvent dans le tableau 1. La figure 1 correspond également à ces résultats sous la forme plus classique d'un *cactus plot*, dans lequel nous n'avons pas dessiné toutes les stratégies, par souci de clarté. La ligne *IND* du tableau correspond au nombre de problèmes pour lesquels le résultat reste *indéterminé* après le temps imparti ou lorsque la mémoire nécessaire dépasse le seuil autorisé. *CEX* indique le nombre de modèles pour lesquels notre outil a trouvé un contre-exemple. La ligne *PRO* représente le nombre de modèles pour lesquels la propriété a été prouvée. La ligne *PRF* correspond à la moyenne de la somme des profondeurs atteintes pour les 178 problèmes les plus difficiles, i.e. qui ont atteint la limite de temps sans dépasser une profondeur de 100. Il s'agit là d'une alternative au score utilisé pour la *deep track* lors de la compétition, qui nous paraît plus significative [5]. Enfin, la ligne *TPS* est la moyenne de la somme des temps nécessaires pour résoudre 92 problèmes non triviaux. C'est-à-dire ceux pour lesquels le temps d'exécution est supérieur à 10 secondes et qui ont été résolus par toutes les stratégies, afin de n'en désavantager aucune.

Analyse des résultats

Tout d'abord, on observe une détérioration des performances et du nombre d'instances résolues lorsque la duplication est effectuée pour toutes les clauses. Que celle-ci soit effectuée dans les bases de données des clauses originales ou apprises. Ceci même en restreignant la duplication au niveau du model checker et en n'infligeant aucune pénalité au solveur avec la gestion de la duplication. On peut également noter que la suppression aggressive a tout de même un impact positif lorsque toutes les clauses sont dupliquées.
Ensuite, on remarque que si on limite la duplication aux clauses de taille 5, quelque soit la base de données

5. A noter que l'ordre est semblable avec les scores utilisés pendant la compétition.

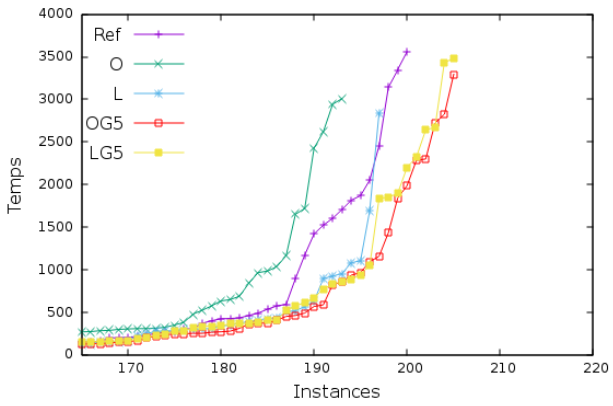


FIGURE 1 – Cactus plot – Nombre d'instances résolues par les différentes stratégies.

dans laquelle on duplique ces clauses, les performances sont meilleures en terme d'instances résolues, de profondeurs moyennes, et de temps moyens.
Enfin, si on ajoute la duplication des glues clauses, le nombre d'instances résolues augmente encore légèrement. De plus, les profondeurs et temps moyens sont encore meilleurs, lorsque l'on effectue la duplication (des glues clauses et des clauses de taille maximale 5) dans la base originale, ce qui est relativement inattendu. Cela signifie probablement que certaines clauses très utiles, ayant un LBD supérieur à 2, sont éliminées lors de la suppression des clauses apprises.
On peut donc en conclure que la meilleure stratégie à adopter consiste à appliquer une sélection drastique des clauses que l'on souhaite dupliquées et de conserver celles-ci indéfiniment, plutôt qu'opter pour une stratégie où plus de clauses sont dupliquées mais où elles risquent d'être supprimées.

6 Conclusion

Dans cet article, nous étendons la duplication pour l'algorithme ZigZag, i.e. une combinaison de BMC et k-induction dans un seul solveur SAT, en considérant les contraintes de chemin simple à la demande qui sont nécessaires pour la complétude de l'algorithme. Nous proposons une solution simple pour la détection des clauses duplicables, très facile à mettre en œuvre. Cette solution a l'avantage de n'avoir aucun impact préjudiciable sur les performances du solveur SAT. Nous montrons également qu'en sélectionnant les clauses à dupliquer à l'aide du score LBD et de leurs tailles, nous sommes capable d'améliorer les performances de notre model checker.
En outre, des améliorations sont encore possibles, notamment : la mise en place d'une politique particu-

lière de suppression des clauses dupliquées, basée par exemple sur leur activité. D'autres critères pourraient également être utilisés pour discriminer les clauses à dupliquer, e.g. SBR (*Size-Bounded Randomized*, [10]). Nous pensons également à étendre la duplication à la méthode d'interpolation ou encore essayer de dupliquer les scores d'activités des variables.

Références

[1] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *IJCAI*, volume 9, pages 399–404, 2009.

[2] Gilles Audemard and Laurent Simon. The glucose sat solver, 2013.

[3] Armin Biere. The aiger and-inverter graph (aig) format. *Available at fmv. jku. at/aiger*, 2007.

[4] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 193–207. Springer, 1999.

[5] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

[6] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397, 1962.

[7] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3) :201–215, 1960.

[8] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[9] Niklas Eén and Niklas Sörensson. Temporal induction by incremental sat solving. *Electronic Notes in Theoretical Computer Science*, 89(4) :543–560, 2003.

[10] Saïd Jabbour, Jerry Lonlac, Lakhdar Sais, and Yakoub Salhi. Revisiting the learned clauses database reduction strategies. *arXiv preprint arXiv :1402.1956*, 2014.

[11] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[12] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a sat-solver. In *International conference on formal methods in computer-aided design*, pages 127–144. Springer, 2000.

[13] Ofer Shtrichman. Tuning sat checkers for bounded model checking. In *International Conference on Computer Aided Verification*, pages 480–494. Springer, 2000.

[14] Ofer Shtrichman. Pruning techniques for the sat-based bounded model checking problem. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 58–70. Springer, 2001.

[15] João P Marques Silva and Karem A Sakallah. Grasp—a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227. IEEE Computer Society, 1997.

[16] Ofer Strichman. Accelerating bounded model checking of safety properties. *Formal Methods in System Design*, 24(1) :5–24, 2004.

Recherche d'heuristique d'équilibre de Nash: quelques résultats préliminaires pour les Constraint Games

Anthony Palmieri **Arnaud Lallouet**

Huawei Technologies Ltd, French Research Center
GREYC, Université de Caen - Normandie
{anthony.palmieri,arnaud.lallouet}@huawei.com

Résumé

Les Constraint Games définissent un cadre pour exprimer et résoudre des jeux statiques à l'aide de la programmation par contraintes. Ils fournissent une représentation compacte et donc peuvent exprimer des jeux plus conséquents. Dans cet article, nous étudions des heuristiques de choix de variable et de valeur spécifique aux Constraint Games. Nous montrons à travers nos expérimentations que les stratégies habituellement utilisées en programmation par contraintes peuvent être amélioré en créant de nouvelles heuristiques dédiés pour les jeux.

1 Introduction

La théorie des jeux [3] est un paradigme pour représenter des interactions entre plusieurs agents (ou joueurs) pouvant avoir des objectifs conflictuels. Ces derniers doivent choisir simultanément une action à réaliser (ou stratégie), pour obtenir une récompense (ou utilité). On peut noter que l'utilisation d'une fonction d'utilité est la façon classique de définir une préférence en théorie des jeux. Le but de chaque agent est de maximiser son utilité en réalisant une de ses actions possibles. Lorsqu'aucun agent n'est capable d'augmenter son utilité en changeant sa stratégie de manière unilatérale, un *équilibre de Nash* est atteint. Ce concept est l'un des plus fondamental en théorie des jeux et a notamment été utilisé avec succès dans de nombreuses situations telles que dans les télécommunications ou en économie. Généralement, les utilités et les stratégies des joueurs sont représentées à l'aide de matrices. Malheureusement, la taille d'une telle représentation croit exponentiellement, rendant impossible la représentation de jeux importants. Ce challenge est d'autant plus important avec l'apparition de nouveaux besoins en théorie des jeux dans certaines applications mobile telle que Waze [1]. Cette dernière permet de diriger efficacement ses utilisateurs sur la route en fonction de leurs préférences et du trafic en temps réel. Waze utilise la théorie des jeux pour guider au mieux ses utilisateurs et plus précisément tente de se rapprocher d'une situation d'équilibre maximisant leur utilité afin d'éviter à ses utilisateurs de pouvoir regretter le choix de route donné par l'application. Ces besoins ont motivé l'introduction de représentation plus compacte telles que les Graphical Games [9], les Boolean Games [6], Actions Graph Games [8], ou plus récemment les Constraints Games [13, 12, 14]. Ces derniers fournissent une représentation compacte des jeux en modélisant les préférences des joueurs à l'aide de la programmation par contraintes. Cependant, même si la représentation de jeux important est maintenant possible, trouver un équilibre de Nash reste NP-Complet [4] dans le cas général et Σ_2^P-complet pour les Boolean Games ou les Constraint Games. Malgré cela, dans certaines applications en temps réel, il est intéressant de trouver rapidement un premier équilibre de Nash. Pour cela, l'étude des heuristiques a potentiellement un impact important. L'organisation de cet article est la suivante : après avoir rappelé les pré-requis, nous nous intéresserons à l'étude d'heuristiques dédiées aux Constraint Games, nous évaluons ensuite les différentes méthodes et enfin nous discuterons des perspectives.

2 Théorie des jeux

Dans cet article, nous nous intéressons seulement aux jeux statiques à information parfaite et complète. Un *jeu* est un triplet $G = (\mathcal{P}, A, u)$ avec $\mathcal{P}$ un ensemble fini de joueurs. L'ensemble $A = (A_i)_{i \in \mathcal{P}}$ où $A_i \neq \emptyset$ cor-

respond à l'ensemble des actions pouvant être jouées par le joueur i. Nous appelons *stratégie* l'action choisi par le joueur i et *profil de stratégies* un tuple $s = (s_i)_{i\in\mathcal{P}}$ où $s_i \in A_i$ correspondant à l'ensemble des actions conjointes des joueurs. L'ensemble des profils de stratégies est noté $A^{\mathcal{P}}$. Les fonctions d'utilité sont données par $u = (u_i)_{i\in\mathcal{P}}$ où $u_i : A^{\mathcal{P}} \to \mathbb{R}$ est l'utilité du joueur i. On note par s_{-i} un profil de stratégies pour tous les joueurs hormis le joueur i, et par $(s_i, s_{-i}) = s$ le profil de stratégie obtenu par la concaténation de s_i et s_{-i}. Sans perte de généralité, nous supposons que chaque joueur souhaite maximiser son utilité. La représentation standard de la fonction d'utilité d'un joueur correspond à une matrice à n dimensions qui définit en extension les valeurs de la fonction d'utilité du joueur.

Example 1 (Location Game)**.** *Ce jeu est variante de [7]. Deux vendeurs de glace désirent choisir un emplacement sur une plage pour installer un stand. Chaque stand peut être placé sur un emplacement numéroté de 1 à 3. On note $e_i \in \{1, 2, 3\}$ la variable de décision du joueur i correspondant à son emplacement choisi. Chaque vendeur i a fixé le prix d'une glace à p_i et on suppose qu'il y a un client à chaque emplacement. Les clients choisissent le vendeur minimisant l'addition de la distance plus le coût d'achat d'une glace. Pour faciliter la lecture de cet exemple, nous fixons le prix de tous les vendeurs à 0.*

Lorsque deux vendeurs sont au même endroit alors ils se partagent à part égale la récompense. Ce jeu peut être traduit sous forme matricielle comme dans la figure 1 où le premier nombre correspond à l'utilité du joueur 1 et le second au joueur 2. On peut noter que les matrices des joueurs sont identiques. La représentation complète d'un tel jeu est contenu dans $2 \times 3^2 = 18$ entiers. Le Location game peut facilement être étendu à n joueurs.

Le concept élémentaire de solution dans les jeux statiques est appelé *Équilibre de Nash en Stratégies Pure* (PNE) et correspond à une situation où aucun joueur n'a intérêt à changer sa stratégie unilatéralement. Une *meilleure réponse* d'un joueur i correspond à l'action du joueur donnant l'utilité maximale connaissant l'action des autres joueurs qui eux ont déjà fixé leur stratégie. En d'autres termes un équilibre de Nash est une situation dans laquelle chaque joueur a répondu avec une meilleure réponse. La figure 2 donne un exemple de déviation du joueur 1 vers sa meilleure réponse. Par exemple, sachant que le deuxième joueur a mis son stand à l'emplacement numéro 3, le premier joueur peut augmenter son utilité et atteindre une

	$e_1 = 1$	$e_1 = 2$	$e_1 = 3$
$e_2 = 1$	$(\frac{3}{2}, \frac{3}{2})$	$(2, 1)$	$(\frac{3}{2}, \frac{3}{2})$
$e_2 = 2$	$(1, 2)$	$(\frac{3}{2}, \frac{3}{2})$	$(1, 2)$
$e_2 = 3$	$(\frac{3}{2}, \frac{3}{2})$	$(2, 1)$	$(\frac{3}{2}, \frac{3}{2})$

FIGURE 1 – Utilité des joueurs sous forme normale dans le jeu du Location Game

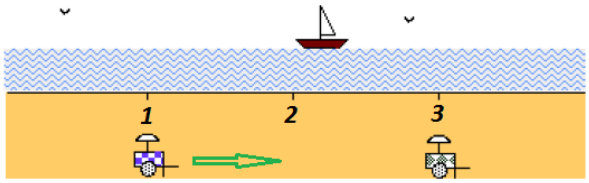


FIGURE 2 – Exemple de déviation dans le Location game

meilleure réponse en se déplaçant vers le second emplacement.

Le jeu du Location Game a 1 PNE. Celui-ci se produit lorsque les joueurs choisissent tous les deux le deuxième emplacement ($e_1 = 2, e_2 = 1$). On voit que celui-ci ne correpond ni à la meilleure utilité possible pour le joueur 1, ni pour le joueur 2.

3 Constraint Games

Les Constraint Games [12, 14] permettent de représenter les jeux de manière compacte en utilisant la programmation par contraintes afin de représenter les fonctions d'utilité des joueurs. Dans les Constraint Games, les stratégies d'un joueur sont représentées à l'aide des variables qu'il possède et son utilité par une condition d'optimisation. Un *Constraint Game* est un 5-uplet $(\mathcal{P}, V, D, G, opt)$ où $\mathcal{P}$ est un ensemble fini de joueurs, V un ensemble fini de variables composé d'une famille d'ensembles disjoint $(V_i)_{i\in\mathcal{P}}$ pour chaque joueur et d'un ensemble V_E de variables *existentielles* disjoint des variables des autres joueurs. Le domaine D est défini comme pour un CSP, et $G = (G_i)_{i\in\mathcal{P}}$ est une famille de CSP sur V représentant *l'objectif* de chaque joueur devant être satisfait en maximisant leur condition d'optimisation opt_i, $\forall i \in \mathcal{P}$. Le couple (objectif, condition d'optimisation) définit l'utilité.

Les Constraint Games permettent de représenter facilement les *contraintes dures* qui définissent des situations qui sont globalement impossibles ou interdites [16]. Pour cela, on ajoute un CSP C au 5-uplet définissant un Constraint Game. Les contraintes dures ont un intérêt lors de la modélisation d'un problème, par exemple dans le Location Game, nous pouvons modéliser le fait que les deux vendeurs ne peuvent pas occuper le même emplacement en ajoutant la contrainte $e_1 \neq e_2$. Les PNE et les meilleures réponses sont seulement cherchés dans l'espace satisfaisant les contraintes dures.

Une variable $x \in V_i$ est dite *contrôlée* par le joueur i. En ce qui concerne les variables existentielles, nous ne les considérons dans cet article que fonctionnellement déterminée par une affectation des variables de décision. Ainsi l'état d'un problème est complètement déterminé par l'affectation totale des variables de décision (et en particulier la valeur de l'objectif est connue). Quand un profil s satis-

fait l'objectif G_i d'un joueur i, l'utilité u_i de ce joueur est donnée par la valeur de la variable devant être maximisé : $u_i(s) = opt_i$. Si un profil s ne peut satisfaire l'objectif G_i, alors ce profil n'est jamais préféré par le joueur i. Dans ce cas, il est équivalent de considérer que la valeur de l'utilité associée est $u_i(s) = -\infty$.

Example 2 (Suite du Location Game)**.** *Nous pouvons définir le Location Game à n joueurs par un Constraint Game.*

- l_i *: définit l'emplacement choisi par le joueur i.*
- $cost_{ic}$ *: définit le coût que le client c doit payer s'il choisit le vendeur i.*
- min_c *: définit le coût minimal que le client c doit payer pour sa glace.*
- $choice_{ic}$ *: variable booléenne qui vaut 1 si le client c choisit le vendeur i.*
- $benefit_i$ *: définit le bénéfice du vendeur i.*
- $nbMin_c$ *définit le nombre de vendeur ayant un coût minimal.*

Le Location Game peut facilement être modélisé par un Constraint Game où chaque vendeur veut maximiser son profit.

- $\mathcal{P} = \{1, \ldots, n\}$
- $\forall i \in \mathcal{P}, V_i = \{l_i\}$
- $\forall i \in \mathcal{P}, D(l_i) = \{1, \ldots, m\}$
- *les contraintes dures C sont les suivantes :*
  - *les vendeurs occupent des emplacements différents :* all_different$(l_1, l_2, \ldots, l_n)$
  - $\forall i \in \mathcal{P}, \forall c \in [1..m]$, $cost_{ic} = |c - l_i| + p_i$
  - $\forall c \in [1..m]$, $min_c = \min(cost_{1c}, \ldots, cost_{nc})$
  - $\forall c \in [1..m]$, $choice_{ic} = 1 \leftrightarrow min_c = cost_{ic}$
  - $\forall c \in [1..m]$, $nbMin_c = \sum_{i=1}^{n} choice_{ic}$
- $\forall i \in \mathcal{P}$, G_i *contient la contrainte suivante :* $benefit_i = p_i . \sum_{c \in [1..m]} \frac{choice_{ic}}{nbMin_c}$
- $\forall i \in \mathcal{P}$, *le critère d'optimisation est* $Opt_i = \max(benefit_i)$

Dans les Constraint Games, les notions de meilleure réponse et de PNE restent inchangées par rapport à ceux des jeux définis sous forme matricielle. De plus nous rappelons que déterminer si un jeu possède un PNE dans un Constraint Game est Σ_2^P-complet [12].

Peu d'algorithmes ont été proposé pour trouver l'ensemble des PNE. Le solveur Gambit [10] propose une procédure d'énumération pour trouver les PNE appelée *enumpure*. Cette procédure vérifie simplement si tous les profils de stratégies sont ou non des PNE. Le principal intérêt de cet algorithme est qu'il fournit une recherche complète et donne en sortie tous les équilibres. Cet algorithme n'a été amélioré que récemment par l'algorithme *Conga* [12, 14] pour les Constraint Games. Conga 1.0 [12] est un algorithme par recherche arborescente mémorisant les meilleures réponses déjà trouvées afin de les réutiliser pour réduire l'espace de recherche. La version 2.0 [14] améliore l'algorithme initial en définissant les préférences comme des contraintes globales et en proposant un mécanisme de propagation. D'autres algorithmes ont été proposés pour un cadre similaire à celui des Constraint Games appelé Asymetric DCOP [5, 17], mais aucun ne permet de modéliser les contraintes dures.

Cependant, même si la recherche de toutes les solutions a été améliorée, aucune nouvelle méthode n'a été proposée pour la recherche d'un premier équilibre.

4 Heuristique de recherche dédiée pour les jeux

Les heuristiques de recherche définissent l'ordre de sélection des variables et des valeurs lors de la résolution d'un problème. Ces dernières ont un impact important tant sur la recherche d'une première solution que la résolution complète d'un problème puisqu'elles guident l'exploration de l'arbre de recherche. Beaucoup de travaux ont été effectués sur les heuristiques en Programmation par Contraintes [2, 11]. Cependant, elles ont été proposé pour tenir compte de la satisfiabilité des contraintes et non des spécificités liées aux jeux. Nous proposons ici une heuristique de recherche adaptée à la recherche d'un premier équilibre de Nash dans les Constraints Games.

L'idée est donc de proposer une heuristique évitant le plus possible aux joueurs d'avoir à regretter leurs choix et donc de vouloir changer leurs stratégies pour une meilleure réponse. Néanmoins, analyser le regret d'un joueur tout au long du parcours de l'arbre de recherche est difficile car pour connaitre la valeur exacte de celui-ci, il faudrait obtenir une affectation complète des variables des autres joueurs et ensuite effectuer un parcours complet sur les variables du joueur afin de déterminer la différence d'objectif pour le choix de chaque stratégie. Effectuer une telle opération reviendrait à résoudre le jeu en entier. C'est pourquoi nous proposons de mesurer l'impact du choix du joueur par l'augmentation de la borne minimale de l'utilité associée à l'affectation de ses variables de décision. En effet, meilleure sera l'utilité associée à la variable de décision du joueur, moins celui-ci aura de chance d'obtenir de meilleure réponse. Plus formellement pour un joueur i, on définit l'impact associé à l'affectation d'une de ses variables de décision x_j par :

$$\sigma_{i/x_j=k} = Obj_{i_{LB}^{after}} - Obj_{i_{LB}^{before}}$$

Avec $Obj_{i_{LB}^{before}}$ et $Obj_{i_{LB}^{after}}$ correspondant respectivement à la borne minimale i avant et après l'affectation de la variable x à la valeur k. Pour chaque joueur, nous mesurons son regret global pour chaque variable et pour chaque valeur. Nous déterminons l'impact du regret pour une variable par la somme de toutes ses affectations et choisissons la variable de regret maximal. Pour éviter d'obtenir une somme strictement croissante et donc de choisir toujours la même variable au même noeud de recherche, nous ajoutons un facteur de vieillissement à la manière de [11].

4.1 Evaluation expérimentale

Pour effectuer l'évaluation expérimentale, nous avons implanté un solveur pour les Constraint Games basé sur Choco 4 [15]. Celui-ci permet d'utiliser plusieurs variables de décision par joueur, de réutiliser toutes les contraintes globales de Choco et réutilise directement la recherche de Choco. Nous comparons notre heuristique *Game Impacts* (GI) avec des stratégies bien connues en Programmation par Contraintes telles que Lexicographic Order avec un choix de valeur à la borne minimale (L_{LB}), à la borne maximale (L_{UB}), Activity (A), Dom/wdeg (DW). Nous avons exécuté les expérimentations sur un serveur Linux Intel Xeon e5-2690, disposant de 256Go de RAM. Pour évaluer notre méthode, nous avons utilisé le jeu du Location Game en 2 dimensions. Dans celui-ci, les joueurs doivent simplement choisir la position de leur emplacement à l'aide de deux coordonnées. Un client ira vers le vendeur le plus proche selon la distance euclidienne.

Game	Param.	L_{LB}	L_{UB}	A	DW	GI
LG	4.4.4	1.36	1.36	**0.14**	1.67	0.33
	5.5.5	163.34	166.02	22.26	890.70	**6.45**
	6.6.6	–	–	–	–	**661.71**

TABLE 1 – Temps d'exécution des différentes heuristiques sur le Location Game en 2 dimensions

Le tableau 1 décrit les temps des différentes méthodes en secondes. Lorsqu'une méthode dépasse un temps d'exécution de 15 minutes, elle est arrêtée (ceci est indiqué par –). La colonne paramètre décrit les paramètres du jeu. Le premier nombre donne le nombre de joueurs et les deux suivants les dimensions X et Y du plan considéré. Les résultats obtenus sont encourageant. En effet, notre nouvelle heuristique permet d'obtenir plus rapidement un premier équilibre de Nash que les stratégies habituellement utilisées en CP grâce à sa spécificité lié aux jeux et plus particulièrement à la recherche d'équilibre de Nash. Cependant, ces résultats doivent être confirmés avec plus d'expérimentations sur divers types de problèmes.

5 Discussion

Dans cet article, nous avons présenté un travail préliminaire sur l'étude d'heuristiques dédiées à la recherche d'un premier équilibre de Nash dans les Constraint Games. Nous avons montré que notre nouvelle heuristique permettait d'obtenir plus rapidement un premier équilibre de Nash en prenant en compte l'évolution des objectifs des joueurs tout au long de la recherche. Les résultats sont plutôt encourageant, mais nécessitent d'être approfondis avec plus d'expérimentations.

Références

[1] Waze - outsmarting traffic, together. `http://www.waze.com/`.

[2] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *European Conference on Artificial Intelligence*, pages 146–150, 2004.

[3] D. Fudenberg and J. Tirole. *Game Theory*. The MIT Press, 1991.

[4] Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Pure nash equilibria : Hard and easy games. *J. Artif. Intell. Res. (JAIR)*, 24 :357–406, 2005.

[5] Tal Grinshpoun, Alon Grubshtein, Roie Zivan, Arnon Netzer, and Amnon Meisels. Asymmetric distributed constraint optimization problems. *J. Artif. Intell. Res. (JAIR)*, 47 :613–647, 2013.

[6] Paul Harrenstein and al. Boolean Games. In Johan van Benthem, editor, *TARK*, 2001.

[7] Harold Hotelling. Stability in competition. In *The Collected Economics Articles of Harold Hotelling*, pages 50–63. Springer, 1990.

[8] Albert Xin Jiang, Kevin Leyton-Brown, and Navin A. R. Bhat. Action-graph games. *Games and Economic Behavior*, 71(1) :141–173, 2011.

[9] Michael J. Kearns, Michael L. Littman, and Satinder P. Singh. Graphical models for game theory. In *UAI*, pages 253–260, 2001.

[10] Richard D McKelvey, Andrew M McLennan, and Theodore L Turocy. *Gambit : Software tools for game theory*, version 16.0.0 edition, 2016.

[11] Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *CP-AI-OR*, pages 228–243. Springer, 2012.

[12] Thi-Van-Anh Nguyen and Arnaud Lallouet. A complete solver for constraint games. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming*. Springer, 2014.

[13] Thi-Van-Anh Nguyen, Arnaud Lallouet, and Lucas Bordeaux. Constraint games : Framework and local search solver. In Éric Grégoire and Bertrand Mazure, editors, *ICTAI*, pages 963–970. Springer, 2013.

[14] Anthony Palmieri and Arnaud Lallouet. Constraint games revisited. In Carles Sierra, editor, *IJCAI, International Joint Conference on Artificial Intelligence*, page to appear, 2017.

[15] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2017.

[16] J. B. Rosen. Existence and uniqueness of equilibrium points for concave n-person games. *Econometrica*, 33(3) :520–534, July 1965.

[17] Mohamed Wahbi and Kenneth N. Brown. A distributed asynchronous solver for nash equilibria in hypergraphical games. In *European Conference on Artificial Intelligence*, pages 1291–1299, 2016.

Une approche à voisinage pour le problème de positionnement d'antennes dans les réseaux cellulaires

Benmezal Larbi[1] *† **Boughaci Dalila**[1] **Benhamou Belaid** [2]

[1] LRIA, USTHB Alger 16111, Algérie
[2] LSIS, AMU Domaine universitaire de Saint Jérôme Avenue
Escadrille Normandie Niemen 13397 Marseille cedex 20, France
larbi07@hotmail.fr dboughaci@usthb.dz belaid.benhamou@univ-amu.fr

Résumé

Le problème de positionnement d'antennes dans les réseaux cellulaires est un problème connu dans le domaine de la télécommunication. Il consiste à choisir parmi un ensemble de sites candidats, les meilleurs emplacements pour installer les stations de bases de façon à maximiser la couverture réseau, tout en minimisant le nombre de stations employées. En théorie, le problème est NP-difficile. Pour le résoudre dans la pratique, nous proposons dans ce travail, l'adaptation de deux métaheuristiques basées sur la recherche locale : Iterated Local Search (ILS)et Breakout Local Search (BLS) et un nouvel algorithme inspiré de l'algorithme ILS. Ce dernier se distingue par un mécanisme de réinitialisation de la recherche et par sa façon de générer une nouvelle solution de départ. Pour valider notre approche, nous avons implémenté, testé et comparé ces algorithmes par rapport à plusieurs autres méthodes sur une instance réelle du problème. Les résultats expérimentaux obtenus montrent que l'approche proposée améliore les performances de ces méthodes.

Abstract

The Antenna Positioning Problem is one of the best-known problem in the field of telecommunication. It consists of maximising the covered area by using a minimum set of base stations. This is known to be an Np-hard optimization problem. To tackle the issue, we used three algorithms : Iterated Local Search (ILS), Breakout Local Search (BLS) and a new algorithm that we propose. The latter was inspired from the algorithm ILS. The new algorithm proposed is based on local search and perturbation to explore the search space and it is characterized by its initialization mechanism. Experiments have been carried out on a realistic benchmark, the results show the efficiency of the proposed algorithm.

1 Introduction

Avec l'avancée technologique importante, le besoin des services offerts par les opérateurs de télécommunication ne cesse d'augmenter. En effet, ces services occupent le centre de nos habitudes quotidiennes. Avec ces besoins croissants, ces opérateurs se doivent d'offrir un réseau capable de les couvrir. D'autre part, le caractère économique de ces fournisseurs de services exige la diminution des coûts d'installation.

La bonne planification des réseaux cellulaires est l'un des moyens permettant de satisfaire ces exigences. En effet, elle permet d'une part, d'assurer un réseau performant, et d'autre part, d'optimiser l'utilisation des ressources, généralement limitées. La planification des réseaux comprend deux principaux problèmes : le problème de positionnement d'antennes ou APP (Antenna Positionning Problem) [5][11] et le problème d'allocation de fréquences ou FAP (Frequency Assignment Problem) [6][9].

On s'intéresse dans ce papier au problème de positionnement d'antennes. Ce dernier englobe un ensemble de décisions concernant le déploiement des ressources physiques dans les réseaux cellulaires. Pour optimiser le réseau, il faudrait minimiser le nombre d'antennes utilisées tout en veillant aux performances du réseau. C'est à dire, installer un nombre minimal d'antennes à des emplacements judicieusement choisis de

*Papier doctorant : Benmezal Larbi[1] est auteur principal.
†Doctorant en co-tutelle, soutenu par le projet PHC-TASSILI 17MDU987

façon à garantir une couverture réseau maximale et un bruit minimal. Le problème peut s'étendre aussi à la spécification du type d'antennes et leur paramétrage (Tilt, Puissance, Azimut).

Le problème est de nature multi-objectif, c'est-à-dire, on cherche une, ou des solutions qui optimisent plusieurs objectifs en même temps. Par contre, il peut aussi être réduit à un problème mono-objectif, dans ce cas, une fonction objectif doit être conçue. Dans notre travail nous traiterons seulement deux objectifs : maximiser la zone couverte par le réseau, et minimiser le nombre d'antennes employées. Ces deux objectifs représentent le centre d'intérêt du problème de positionnement d'antennes dans les réseaux cellulaires. La difficulté réside dans le fait que ces deux objectifs sont difficiles à satisfaire en même temps. En plus, le problème est caractérisé aussi par le nombre important de solutions possibles. Le problème à résoudre est un problème d'optimisation combinatoire NP-Difficile.

Plusieurs travaux de recherche ont abordé le problème et plusieurs méthodes ont été proposées pour le résoudre dans la pratique. Ces dernières sont généralement des méthodes approchées ; c'est les plus adaptées à ce type de problème. Les méthodes exactes ne peuvent résoudre que des instances de petites tailles. On ne cherche pas à trouver une solution exacte, mais on cherche plutôt une solution de bonne qualité dans un temps raisonnable.

Dans ce travail, nous présentons trois métaheuristiques de voisinage pour résoudre le problème de positionnement d'antennes dans les réseaux cellulaires. Tout d'abord, nous adaptons les algorithmes ILS (Iterated Local search) [10] et BLS (Breakout Local Search)[4] pour le problème étudié. Ensuite, nous présentons un nouvel algorithme que nous proposons. Ces algorithmes emploient la recherche locale comme mécanisme d'intensification de la recherche dans la zone en cours d'exploration, et pour sortir d'un optimum local, ils utilisent des mécanismes de perturbation. Cela leur permet de continuer la recherche à partir d'un autre point, et explorer divers zones dans l'espace de recherche.

Ce papier est organisé comme suit : Dans la section suivante, nous présentons les différents travaux qui ont été réalisés sur le problème. Nous présentons dans la section 3, une modélisation du problème basée sur les hypergraphes, proposée par Mendes et al. Dans la section 4, nous présenteons les deux adaptations des deux algorithmes de résolution ILS et BLS, à la suite desquels nous décrivons le nouvel algorithme que nous proposons. Nous réalisons dans la section 5 une étude expérimentale où nous présentons les résultats obtenus lorsque ces algorithmes sont appliqués sur une instance réelle du problème. Les conclusions et perspectives sont données dans la dernière section.

2 Travaux connexes

Plusieurs travaux ont été réalisés dans le domaine. On peut citer comme exemple les travaux de Reininger [14] et Reininger et Caminada [15], où, une nouvelle modélisation du problème a été proposée. Dans cette dernière, plusieurs fonctions qui représentent divers objectifs et contraintes du problème sont définies, comme la couverture réseau, le bruit, le trafic pris en charge, le nombre d'antennes utilisées et d'autres. Ces fonctions doivent être optimisées. Cette modélisation reflète bien la nature multi-objectif du problème. Plusieurs autres travaux ont utilisé cette modélisation pour traiter le problème, comme dans Meunier et al.[12], où ils ont utilisé un algorithme génétique multi-objectif lors de la phase de résolution, et dans Vasquez et al. [13] où ils ont proposé une approche en trois phases, basée sur la recherche taboue. Une première phase de prétraitement, vise à éliminer les solutions qui utilisent un nombre d'antennes, soit très petit, soit très grand. La deuxième phase est une phase de recherche, et la troisième a pour but le paramétrage des antennes. El-Ghazali et al. [7] se sont aussi basés sur cette modélisation pour proposer des modèles parallèles hiérarchiques. Ils ont montré que l'utilisation de ces derniers peut accélérer la recherche et permet aussi de trouver des solutions de très bonne qualité. Un algorithme génétique a été utilisé pour la résolution du problème.

La modélisation de Reininger est une modélisation qui reflète bien la nature complexe du problème traité, dans le sens où elle modélise plusieurs aspects. Néanmoins, la résolution d'instances basées sur cette modélisation peut être très difficile.

Une autre modélisation a été définie par Calegari et al. [5]. Contrairement à la modélisation de Reininger et al., cette modélisation fait abstraction à plusieurs aspects techniques du problème, comme les paramètres d'antennes et les stations mobiles. La zone est modélisée par un graphe biparti, où l'on cherche à trouver le plus petit ensemble dominant. Deux objectifs seulement sont pris en compte par cette modélisation : la maximisation de la couverture et la minimisation du nombre d'antennes employées. Ces deux objectifs sont représentés par une seule fonction objectif. Donc, le problème modélisé est un problème mono-objectif.

Plusieurs méthodes ont été appliquées sur des instances basées sur cette modélisation. On cite par exemple, l'algorithme génétique et l'algorithme CHC dans Alba et al. [1,2], et NSGA-II et MOCHC dans Nebro et al. [3]. Ces deux derniers sont une adaptation multi-objectif des algorithmes GA et CHC afin

de leur permettre de traiter plusieurs fonctions objectif en même temps. Pour ça, Ils ont défini leurs propre fonctions à optimiser.

Mendes et al. ont proposé une nouvelle modélisation dans [11]. Cette dernière améliore celle de Calégari en proposant de modéliser la surface géographique par un hyper-graphe au lieu d'un graphe. Une large panoplie d'algorithmes a été testée dans ce travail sur une instance réelle du problème.

Dans ce papier on comparera les performances des algorithmes étudiés avec ceux testés dans le travail de Mendes et al., en les exécutants sur la même instance.

3 Modélisation du problème

Pour représenter le problème, nous utilisons la modélisation de Mendes, décrite dans [11]. Cette modélisation propose un modèle basé sur la théorie des graphes pour définir les différentes composantes du réseau.

3.1 Zone géographique

La zone géographique à couvrir est discrétisée pour avoir en résultat un ensemble fini d'emplacements. Chaque emplacement est représenté par un point caractérisé par ses coordonnés x et y. Parmi ces points, quelques uns sont choisis pour être des sites candidats à héberger des stations de bases. La zone est composée de deux types de points :

- Points de type L : ce sont les points qui doivent être couverts ;
- Points de type M : les points qui représentent les sites candidats pour héberger les stations de base.

La zone est modélisée par un hypergraphe $H(V, E)$ tel que, V est l'ensemble des sommets de l'hypergraphe H, et E est une partie de $P(V)$ (l'ensemble des parties de l'ensemble V). On appelle les éléments de E les hyper-arêtes. Une hyper-arête généralise la notion d'arête, dans le sens où elle permet de relier plusieurs arrêtes au lieu de deux seulement. Dans notre cas, l'ensemble V contient tous les points de la zone géographique ; $V = M \cup L$, et E l'ensemble qui contient toutes les hyper-arêtes qui lient chaque point de type M avec les points couverts par ce dernier. Notre hypergraphe peut être vu comme un ensemble de graphes orientés. Chaque graphe possède un sommet central. Ce dernier est un point de type M (représente une station de base) associé avec des points de l'ensemble V représentant les points qui ce trouvent dans la cellule formée par cette station de base.

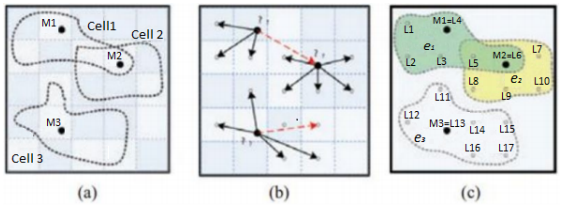


Fig. 1. Exemple de trois transmetteurs représentés par un hypergraphe

La figure 1 illustre un exemple de trois transmetteurs représentés avec leurs cellules associées (Fig.1.a). Ils ont été modélisés par un ensemble de graphes qui associent les transmetteurs avec les points qu'ils couvrent (Fig.1.b). La représentation de l'hypergraphe associée est donnée dans la figure (Fig.1.c). On voit bien la ressemblance entre les cellules formées par les transmetteurs et leur modélisation par l'hypergraphe.

3.2 Antenne

Des antennes dites Isotropes sont utilisées dans cette modélisation ; ce sont des modèles théoriques qui rayonnent uniformément dans toutes les directions. Une seule antenne seulement de ce type est installée dans une station de base.

3.3 Fonction objectif

Les deux objectifs à optimiser sont combinés dans une seule fonction objectif notée $f(x)$. Cette dernière a été définie dans [5].

$$f(x)=\frac{Tauxdecouverture(x)^2}{Nombredetransmetteursutilises(x)}$$

Avec :

$$Taux\ de\ couverture(x)=\frac{Voisin(M',E')}{Voisin(M,E)}$$

tel que
$voisins(M, E) = \{v \in V | \exists u \in M, (u, v) \in E\}$
et M' est l'ensemble des transmetteurs utilisés par la solution x. E' est l'ensemble des hyper-arêtes qui lient les points de l'ensemble M' avec leurs points associés

Le problème à résoudre rappel le problème de couverture par ensembles (Set Covering Problem dans la littérature originale). Il consiste à trouver un sous ensemble de sommets de taille minimum qui couvre tout le graphe. Une seule différence réside dans le but des deux problèmes, là on ne cherche pas à assurer une couverture totale de la zone, mais on cherche plutôt à la maximiser. Le problème est connu pour être NP-Difficile

3.4 Solution

Une solution au problème indique pour chaque emplacement candidat, si une station de base est installée ou non. Cela peut être modélisé par un vecteur binaire, où chaque bit représente un emplacement candidat. Si une station de base est installée dans un emplacement, son bit associé est alors mis à 1, sinon il est mis à 0. De ce fait, La taille du vecteur solution est égale au nombre d'emplacements candidats.

4 Algorithmes d'optimisation proposés

Nous proposons de résoudre le problème de positionnement d'antennes par trois algorithmes : l'adaptation de la méthode ILS (Iterated Local Search) de Lourenço et al. [10], celle de l'algorithme BLS (Breakout Local Search) de Benlic et al. [4] et un nouvel algorithme que nous proposons.

Ces algorithmes sont des algorithmes de voisinage qui utilisent la recherche locale comme moyen d'intensification. D'où la nécessité de définir un voisinage d'une solution donnée.

Définition Un voisinage N est une fonction $N : S \rightarrow P(S)$, tel que S représente l'espace de recherche et $P(S)$ l'ensemble des parties de S. Le voisinage d'une solution $s \in S$ est l'ensemble des solutions $N(s)$ de $P(S)$ qui peuvent être obtenues en effectuant un seul mouvement à partir de s.

Nous avons défini un mouvement guidé par une propriété des réseaux cellulaire. Cette propriété est basée sur le fait qu'une bonne solution ne possède généralement pas beaucoup d'antennes installées dans un périmètre trop proche.

Définition Un mouvement à partir d'une solution $s \in S$ sélectionne une station de base non active pour l'activer, et désactive les autres stations localisées dans un périmètre de rayon r.

4.1 La méthode ILS adaptée

L'algorithme ILS applique une recherche locale à partir d'une solution initiale. Cette recherche locale retourne un optimum local, qui n'est pas forcément la meilleure solution qui puisse être, mais d'autres solutions de meilleure qualité peuvent bien encore exister. Pour explorer d'autres zones de l'espace de recherche, ILS applique à chaque fois une perturbation (ligne 5) de la solution pour s'échapper de la zone d'attraction. Pour perturber une solution, on choisit aléatoirement un nombre L de stations de bases et on change leurs états. Ensuite, une recherche locale est appliquée pour tester la qualité d'un autre optimum local.

Algorithm 1 La méthode Iterated Local Search

Require: Une instance du problème
Ensure: La meilleure solution trouvée S_m
1: $S0$ = GenererSolutionInitiale()
2: S = RechercheLocale($S0$)
3: $S_m = S$
4: **while** Condition d'arrêt non satisfaite **do**
5: S' = Perturbation (S, *historique*)
6: S" = RechercheLocale(S')
7: **if** S" est meilleure que S_m **then**
8: S_m=S"
9: **end if**
10: S= CritèreAcceptation(S, S",*historique*)
11: **end while**

ILS réitère le processus à partir d'une solution choisie entre S et S". La fonction CritèreAcceptation (ligne 10) assure le choix entre ces deux solutions. Cette fonction peut avoir différentes implémentations. Par exemple, elle peut être implémentée de façon à choisir toujours la meilleure solution, ou bien de façon à choisir toujours la nouvelle solution obtenue. Plusieurs implémentations intermédiaires sont aussi possibles. Dans ces cas, le paramètre *historique* peut être un moyen pour trouver un équilibre entre l'intensification et la diversification. Par exemple, ce paramètre peut être le numéro de l'itération courante. Dans ce cas, la fonction CritèreAcceptation peut se baser sur ce paramètre pour faire un choix entre les deux solutions.

Dans notre travail nous avons implémenté une fonction qui choisi toujours le meilleur optimum entre ceux rencontrés dans l'itération courante et l'itération précédente. Ainsi, l'algorithme ILS parcourt les optimums locaux répartis dans l'espace de recherche et garde l'historique de la meilleure solution rencontrée.

Après un nombre I d'itérations sans amélioration de la meilleure solution, ILS réinitialise la recherche à partir d'une nouvelle solution de départ générée aléatoirement. Le pseudo-code de la méthode ILS adaptée est donné dans l'algorithme 1.

4.2 La méthode BLS adaptée

Algorithm 2 La méthode Breakout Local Search

Require: Une instance du problème, $L0$: degré du saut initial, Ls : degré du saut lors d'une forte perturbation, T : nombre maximum d'optimums locaux visités sans amélioration de la meilleure solution S_m.
Ensure: La meilleure solution trouvée S_m

```
1: S = GenererSolutionInitiale()
2: S_m =S
3: S_p = S {S_p : la solution optimale obtenue lors de la dernière recherche locale}
4: nbNonAmlioration=0 {Nombre de fois où la solution optimale n'a pas été améliorée}
5: while Condition d'arrêt non satisfaite do
6:    S=RechercheLocale(S)
7:    if S est meilleure que S_m then
8:       S_m =S {m-à-j de la meilleure solution}
9:       nbNonAmlioration=0
10:   else
11:      nbNonAmlioration = nbNonAmlioration+1
12:   end if
13:   if nbNonAmlioration > T then
14:      {la recherche est stagnée, une forte perturbation est nécessaire}
15:      L = Ls
16:   else
17:      if S=S_p {La recherche est retournée vers le précédant optimum, Le degré de perturbation est augmenté} then
18:         L = L + 1
19:      else
20:         L = L0
21:      end if
22:   end if
23:   S_p = S
24:   S=PerturberSolution(S,L,nbNonAmlioration)
25: end while
```

L'algorithme BLS suit le même schéma que celui de l'algorithme ILS, mais possède en plus, un mécanisme de perturbation avec une force paramétrable. A chaque fois que la recherche retourne vers le même optimum, la force de perturbation est augmentée (ligne 18). Après un nombre T d'itérations sans amélioration de la meilleure solution, une forte perturbation est appliquée (ligne 15). Pour perturber une solution, on choisit aléatoirement un nombre L de stations de bases est on change leurs états. Le pseudo-code de la méthode BLS adaptée à notre problème est donné dans l'algorithme 2.

4.3 Le nouvel algorithme proposé

Nous proposons un nouvel algorithme inspiré des deux algorithmes ILS et BLS. Notre algorithme utilise aussi la recherche locale comme moyen d'intensification (ligne 7), et la perturbation de la solution pour apporter une diversification (ligne 22). Notre algorithme possède un mécanisme de réinitialisation (ligne 24)équivalent à la forte perturbation de l'algorithme BLS, mais diffère par son critère de réinitialisation. Le pseudo-code de la méthode est donné dans l'algorithme 3.

Deux variables clés sont utilisées par notre algorithme : S_m et S'. La première, enregistre la meilleure solution trouvée depuis le début de la recherche, et la deuxième, représente la meilleure solution trouvée depuis la dernière réinitialisation de la recherche. A chaque fois que S' est améliorée, elle est comparée avec S_m. Si S' améliore la meilleure solution, alors S_m est mise à jour. Après un certain nombre P d'itérations sans amélioration de S', la force de perturbation est augmentée (ligne 20). Si malgré cette augmentation S' n'a toujours pas été améliorée, et que le nombre d'itérations dépasse un nombre I, une réinitialisation est déclenchée.

4.3.1 Le critère de réinitialisation

Notre algorithme se distingue par rapport à ILS et BLS par son critère de réinitialisation. En effet, ce dernier est guidé par l'état d'évolution de la meilleure solution trouvée depuis la réinitialisation précédente S', et non pas par l'état d'évolution de S_m, la meilleure solution trouvée depuis le début de la recherche,comme dans ILS et BLS (lorsqu'il effectue une forte perturbation).

L'idée de considérer S' au lieu de S_m est basée sur le principe qui dit que, plus la recherche avance, plus la meilleure solution trouvée devient bonne, et devient difficile à améliorer. Alors, si l'on se base sur S_m pour réinitialiser la recherche, les réinitialisations deviendront très fréquentes, et cela permet d'avantager la diversification par rapport à l'intensification de la recherche. Par contre, si l'on considère S' au lieu de S_m, la balance entre l'intensification et la diversification sera plus équilibrée. En effet, on sait que S' est généralement de moins bonne qualité que S_m, ce qui implique que S' possède plus de chances d'être améliorée. Lorsque S' est améliorée, on se dit que le chemin exploré peut mener vers des solutions meilleures, donc la réinitialisation est retardée pour l'explorer d'avantage.

Algorithm 3 Notre méthode

Require: : Une instance du problème, P : nombre maximum d'optimums locaux visités sans amélioration avant la perturbation, $forcePerturbationInitiale$: force de perturbation initiale, I : nombre maximum d'optimums locaux visités sans amélioration avant la réinitialisation.
Ensure: La meilleure solution trouvée S_m

$S = GenererSolutionInitiale()$
$S_m = S$
$S' = S$ {S' est La meilleure solution trouvée depuis la dernière réinitialisation}
$Cpt = 0$ {cpt compte le nombre d'itérations sans amélioration de S'}
$forcePerturbation = forcePerturbationInitiale$
while Condition d'arrêt non satisfaite **do**
  S=RechercheLocale(S)
  if S est meilleure que S' **then**
    $S' = S$
    Cpt=0
    if S' est meilleure que S_m **then**
      $S_m = S'$ {m-à-j de la meilleure solution}
    end if
  else
    $cpt = cpt + 1$
  end if
  if $Cpt < I$ **then**
    if $Cpt > P$ **then**
      {La recherche est stagnée, une augmentation de la force de perturbation est nécessaire}
      $forcePerturbation = forcePerturbation+1$
    end if
    S= Perturber($S, forcePerturbation$)
  else
    S= RéinitialiserSolution(S_m, $nombreEval$) {$nombreEval$ est le nombre de fois où la fonction objectif a été évaluée depuis le lancement de l'algorithme}
    $forcePerturbation = forcePerturbationInitiale$
    Cpt=0
    $S' = S$ {Mise à jour de S'}
  end if
end while

4.3.2 La procédure de réinitialisation

Lorsque la recherche est stagnée, et le chemin exploré ne permet plus l'amélioration de la meilleure solution trouvée, le mécanisme de réinitialisation permet de redémarrer la recherche à partir d'une nouvelle solution de départ. Cette stratégie permet d'explorer de nouvelles zones dans l'espace de recherche. Le pseudocode de la procédure de réinitialisation est donné dans l'algorithme 4.

Algorithm 4 Procédure de réinitialisation

Require: S_m :la meilleure solution trouvée, $nombreEval$:nombre d'évaluations effectué jusque là, $maxEval$: nombre d'évaluations maximum fixé au lancement de notre algorithme.
Ensure: Nouvelle solution S

S = générer une solution aléatoirement
for i=0 to i=$tailleSolution$ **do**
  Soit $rand$ un nombre aléatoire entre 1 et 100
  $stadeAvancement = nombreEval/maxEval * 100$ {$sadeEvancement$ représente l'état d'avancement de la recherche}
  if $rand < stadeAvancement$ **then**
    $S[i] = S_m[i]$
  end if
end for

La procédure de réinitialisation permet de générer des solutions aléatoires combinées avec la meilleure solution trouvée S_m. Cette combinaison est basée sur l'évolution de la recherche. Si la recherche est à un stade initial, la solution générée s'approche alors plus de la solution aléatoire. Mais plus la recherche avance, plus la solution générée s'approche de S_m. Par exemple, si la recherche est à un stade de 20%, alors la nouvelle solution aura en moyenne 20% des éléments de la meilleure solution et 80% de la solution aléatoire. Si l'on est à un stade de 90%, alors la nouvelle solution contiendra en moyenne 90% des éléments de la meilleure solution. Ainsi, le degré de diversification apporté par ce mécanisme diminue au fur et à mesure que la recherche évolue.

5 Résultats expérimentaux

Afin d'évaluer les performances de chaque algorithme, nous les avons exécuté sur une instance réelle fournie dans [8]. Cette instance est une représentation de la zone géographique de la ville de Malaga. Cette zone à une surface de 27,2 km^2 discrétisée par une grille de taille 450 X 300. Ce qui donne au total 135000 points. Chaque point représente une surface de 15 X 15 m^2. Parmi ces points, 1000 sites candidats

sont répartis sur l'ensemble de la grille. Chaque site candidat est défini par sa position (x ,y) dans la grille. Des antennes isotropes de rayon de couverture égal à 30 points ont été utilisées[11]. La zone étudiée possède des endroits dont il est impossible d'installer des antennes. Ces endroits peuvent être, la mer, les montagnes ...etc., ce qui implique que seulement un taux de 95,522% de couverture peut être atteint.

Fig. 2. Image correspondant à la zone géographique à couvrir (ville de Malaga)

5.1 Critère de comparaison

Les trois algorithmes sont comparés par rapport à la valeur de la fonction objectif de la meilleure solution trouvée. Pour que la comparaison soit sur la même base, tous les algorithmes exécutent le même nombre d'évaluations des solutions. Plusieurs séries d'exécutions pour différents nombres d'évaluations ont été effectuées. Pour chaque série de test, 10 exécutions ont été réalisées.

5.2 Paramètres

Plusieurs séries de tests ont été réalisées pour fixer les meilleurs paramètres pour chaque algorithme. La valeur de chaque paramètre est donnée comme suit : ILS : $I = 10, L = 5$; BLS : $L0 = 5, Ls = 10, T = 10$; notre algorithme : $forcePerturbationInitiale = 2; P = 3; I = 10$.

Les résultats obtenus sont représentés dans le graphe ci-dessous. L'axe des x représente le nombre d'évaluations de la fonction objectif et l'axe des y représente la moyenne des valeurs de la fonction objectif obtenues lors des 10 exécutions :

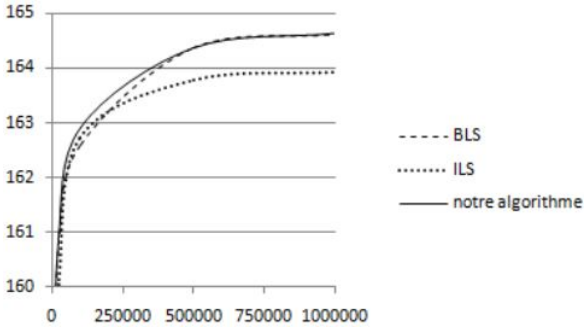


Fig. 3. Valeur moyenne de la fonction objectif obtenue en fonction du nombre d'évaluations.

Tableau 1. Taux de couverture moyen et nombre d'antennes moyen employées par les solutions trouvées par chaque algorithme

Algorithme	Taux de couverture moyen	Nombre d'antennes moyen
Notre algorithme	87.03%	46
BLS	87.02%	46
ILS	86.83%	46

Les moyennes des valeurs de la fonction objectif obtenues lors des dix exécutions, montrent que notre algorithme dépasse en général les algorithmes ILS et BLS qui sont déjà très bien classés par rapport aux résultats connus [11].

Le tableau 1. montre aussi que notre algorithme permet d'atteindre le meilleur taux de couverture pour le même nombre d'antennes employées.

Mendes et al.[11] ont testé une large panoplie d'algorithmes sur le même benchmark que nous avons employé, donc pour valider nos résultats, nous les avons comparé par rapport aux résultats obtenus par ces derniers.

Tableau 2. Résultats de nos trois algorithmes comparés par rapport à la fonction objectif avec ceux testés dans [11].

Algorithme	Valeur moyenne de la fonction objectif
MS_GEPVNS	164.701
Notre algorithme	164.642
BLS	164.608
ILS	163.911
CHC	163.278
PBIL	162.651
HYBRID_RUFNS	162.411
AGC	162.134
MS_VNS	162.120
MS_FNS	161.884
CAPMC	161.727
GRASP_EVNS	161.352
SA	156.478
DE	148.802
GRASP_SRCL	148.196

Nos trois algorithmes sont classés respectivement deuxième, troisième et quatrième devant plusieurs algorithmes testés. Ils ont dépassés plusieurs algorithmes connus dans la littérature comme l'algorithme génétique(GA), le recuit simulé(SA), PBIL et CHC. Ces deux derniers sont des algorithmes évolutionnaires. l'algorithme PBIL est une combinaison entre l'algorithme génétique et l'apprentissage compétitif. L'apprentissage permet de générer les nouveaux individus de la population. Le second algorithme (CHC), utilise les opérations des algorithmes évolutionnaires, comme la sélection et la mutation, toute en se basant sur des opérateurs spécifiques(sélection élitiste et croisement perturbateur)qui permettent de générer toujours une population diversifiée. MS_GEPVNS qui est une version améliorée de l'algorithme VNS est le seul algorithme qui dépasse nos trois algorithmes en termes de qualité de solution.

6 Conclusion

Nous avons présenté dans ce papier trois algorithmes pour la résolution du problème de positionnement d'antennes. Nous avons adapté les algorithmes ILS et BLS à notre problème et nous avons aussi proposé une nouvelles approche inspirée de ces deux algorithmes mais qui diffère par son système de réinitialisation de la recherche.

Ces algorithmes ont été implémentés, testés et comparés par rapport à la valeur moyenne de la fonction objectif des meilleures solutions trouvées pour un benchmark réel représentant la ville de Malaga. Ils ont aussi été comparés avec d'autres algorithmes connus dans la littérature et les résultats obtenus sont très prometteurs. En perspective, nous avons plusieurs directions pour améliorer ce travail. Tout d'abord introduire des optimisations sur le mouvement utilisé en incluant un bruit aléatoire. Nous pouvons aussi introduire ce même principe dans le processus de réinitialisation de la nouvelle méthode. Nous sommes aussi intéressé d'inclure certaines caractéristiques du voisinage de la méthode VNS dans nos méthodes afin d'améliorer leurs efficacités. Enfin, étendre notre étude expérimentale à d'autres instances du problème.

Références

[1] E. Alba, G. Molina, and F. Chicano. Optimal placement of antennae using metaheuristics. In Numerical Methods and Applications (NM&A-2006), Borovets, Bulgaria, August 2006.

[2] Alba, E. and Chicano, F., 2005. On the Behavior of Parallel Genetic Algorithms for Optimal Placement of Antennae in Telecommunications. International Journal of Foundations of Computer Science, 16 (2), 343 – 359.

[3] Nebro, A. J., Alba, E., Molina, G., Chicano, F., Luna, F., & Durillo, J. J. (2007, July). Optimal antenna placement using a new multi-objective CHC algorithm. In Proceedings of the 9th annual conference on Genetic and evolutionary computation (pp. 876-883). ACM.

[4] Benlic U., Hao J.K. : Breakout local search for maximum clique problems. Accepted to Computers & Operations Research, DOI :10.1016/j.cor.2012.06.002 (2012)

[5] Calégari, P., Guidec, F., Kuonen, P., & Wagner, D. A. W. D. (1997, May). Genetic approach to radio network optimization for mobile systems. In Vehicular Technology Conference, 1997, IEEE 47th (Vol. 2, pp. 755-759). IEEE.

[6] D. Castelino , S. Hurley , And N. Stephens, "A Tabu Search Algorithm for frequency assignment", in Annals of Operations Research 63, pp : 301-319, 1996.]

[7] El-Ghazali Talbi, Hervé Meunier, "Hierarchical parallel approach for GSM mobile network design", In J.Parallel Distrib. Comput. 66 (2006) 274 – 290

[8] J. Gómez-Pulido. (2008). Web Site of Net-Centric Optimization [Online]. Available : http ://oplink.unex.es/rnd

[9] Yasmine LAHSINAT – Belaïd BENHAMOU – Dalila BOUGHACI "Trois hyperheuristiques pour

le problème d'affectation de fréquence dans un réseau cellulaire" , Dans les proceedings de JFPC 2015, LABRI, Bordeaux, 22-24, 2015, jui 2015

[10] H. R. Lourenco, O. C. Martin, and T. Stützle, "Iterated local search," in Handbook of Metaheuristics, Boston, MA : Kluwer, 2002, pp. 321–353

[11] Mendes, S.P., Molina, G., Vega-Rodrguez, M.A., Gomez-Pulido, J.A., Sez, Y., Miranda, G., Segura, C., Alba, E., Isasi, P., Len, C., Snchez-Prez, J.M. : Benchmarking a Wide Spectrum of Meta-Heuristic Techniques for the Radio Network Design Problem. IEEE Transactions on Evolutionary Computation, 1133–1150 (2009)

[12] Meunier, H., Talbi, E. G., & Reininger, P. (2000). A multiobjective genetic algorithm for radio network optimization. In Evolutionary Computation, 2000. Proceedings of the 2000 Congress on (Vol. 1, pp. 317-324). IEEE.

[13] Michel Vasquez , Jin-Kao Hao A Heuristic Approach for Antenna Positioning in Cellular Networks. Journal of Heuristics, 7 : 443–472, 2001 Kluwer Academic

[14] Reininger, P. (1997). "ARNO Radio Network Optimisation Problem Modelling," ARNO Deliverable N 1-A 1-Part 1. FT. CNET, July 15, 1997.

[15] Reininger, P. and A. Caminada. (1998a). "Model for GSM Radio Network Optimisation." In 2nd Intl. ACM/IEEE Mobicom Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM), Dallas, December 16, 1998.

Construction et amélioration de strategies SMT

N. Galvez [1,2] **Y. Hamadi**[3] **E. Monfroy**[4] **F. Saubion** [2]

[1] LabDII, UTFSM, Valparaíso, Chili
[2] LERIA, Université d'Angers, France
[3] LIX, École Polytechnique, France
[4] LS2N, UMR CNRS 6241, Université de Nantes, France
ngalvez@inf.utfsm.cl monfroy-e@univ-nantes.fr saubion@info.univ-angers.fr

Résumé

Un défi dans le domaine des problèmes de satisfaction modulo théories (SMT) est de concevoir des outils à la fois théoriques et pratiques pour que les utilisateurs puissent bénéficier et utiliser efficacement un large éventail heuristiques et de mécanismes de résolution au sein d'un même solveur, tel que le solveur [3]. Toutefois, la combinaison et le contrôle de ces composants nécessitent bien souvent une connaissance experte tant des domaines sur lesquels reposent les modèles à résoudre que des performances attendues et de la pertinence de chaque élément opérationnel. Ainsi, étant donné un ensemble d'instances SMT à traiter et un solveur donné, il convient d'écrire une stratégie de résolution qui définit la manière dont les mécanismes de simplification, vérification, ou encore de résolution de sous-problèmes, seront agencés. Cette phase peut être vue comme la conception de la structure de la stratégie. De plus, un certain nombres de paramètres doivent également être ajustés pour assurer un bon comportement et une bonne efficacité du solveur.

Dans cet article, notre objectif est de proposer un cadre de génération automatique de stratégies, suffisamment général pour être utilisé par un utilisateur non averti ou encore pour améliorer des stratégies proposées par des experts. Basée sur la programmation génétique [4], notre approche permet, à partir de squelettes simples de stratégies basiques, de construire et d'ajuster de nouvelles stratégies efficaces, tout en gérant un nombre de paramètres et d'options possibles très important. Sur un ensemble d'instances extraites de la compétition SMT, nous montrons qu'une telle démarche peut permettre d'améliorer les performances des stratégies par défaut du solveur Z3 [2] (développées pour ces compétitions [1]) et de résoudre de nouvelles instances.

Cet article a été présenté lors de la conférence ICTAI 2016 [5].

Références

[1] Clark Barrett, Morgan Deters, Leonardo de Moura, Albert Oliveras, and Aaron Stump. 6 years of SMT-COMP. *Journal of Automated Reasoning*, pages 243–277, 2013.

[2] Leonardo de Moura and Nikolaj Bjørner. Z3 : An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008.*, LNCS, pages 337–340. Springer, 2008.

[3] Leonardo de Moura and Grant Olney Passmore. The Strategy Challenge in SMT Solving. In *Automated Reasoning and Mathematics*, LNCS, pages 15–44. Springer, 2013.

[4] John R. Koza. *Genetic Programming : On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[5] N. Gálvez Ramírez, Y. Hamadi, E. Monfroy, and F. Saubion. Evolving smt strategies. In *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 247–254, Nov 2016.