JFPC 2016

Emmanuel Hébrard

Douzièmes Journées Francophones de Programmation par Contraintes

JFPC 2016

Montpellier, 15-17 juin 2016

Photos by Wolfgang Staudt

JFPC 2016

Douzièmes Journées Francophones de Programmation par Contraintes

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

Organisation

SupAgro, LIRMM, Montpellier

Président des Journées

Philippe Vismara

Président du comité de programme

Emmanuel Hebrard

Comité d'organisation

Président

Philippe Vismara	LIRMM, MISTEA

Vice-présidente

Marie-Laure Mugnier	LIRMM

Membres

Abdallah Arioua	LIRMM
Nicolas Briot	LIRMM
Madalina Croitoru	LIRMM
Rmi Coletta	LIRMM
Abdelraouf Hecham	LIRMM
Galle Hisler	LIRMM, Tellmeplus
Nadjib Lazaar	LIRMM
Michel Leclre	LIRMM
Martine Marco	Montpellier SupAgro, MISTEA
Gilles Trombettoni	LIRMM
Vronique Sals-Vettorel	Montpellier SupAgro, MISTEA
Olivier Sans	LIRMM

Comité de programme

Lucas Bordeaux	Microsoft Research
Hadrien Cambazard	G-SCOP, Université de Grenoble Alpes
Sophie Demassey	CMA, MINES ParisTech
Gilles Dequen	MIS, Université de Picardie Jules Verne
Alban Derrien	LINA, Ecoles des Mines de Nantes
Jean-Guillaume Fages	COSLING S.A.S.
Steven Gay	ICTEAM, Université catholique de Louvain
Carmen Gervet	ESPACE-DEV, Université de Montpellier
Djamal Habet	LSIS, Université d'Aix-Marseille
Marie-José Huguet	LAAS-CNRS, INSA, Université de Toulouse
Jean Marie Lagniez	CRIL, Université d'Artois
Frederic Lardeux	LERIA, Université d'Angers
Nadjib Lazaar	LIRMM, Université de Montpellier
Olivier Lhomme	IBM France
Xavier Lorca	Ecole des Mines de Nantes
Jean-Noël Monette	Tacton Systems AB
Samba Ndojh Ndiaye	LIRIS, Université de Lyon
Alexandre Niveau	GREYC, Université de Caen Normandie
Alexandre Papadopoulos	LIP6, Université Pierre et Marie Curie, Sony CSL
Anastasia Paparrizou	LIRMM, CNRS, Université de Montpellier
Marie Pelleau	LIP6, ENS, Universit de Pierre et Marie Curie
Thierry Petit	Worcester Polytechnic Institute
Jean-Charles Regin	I3S, University of Nice-Sophia Antipolis
Stephanie Roussel	ONERA - the French Aerospace Lab
Mohamed Siala	Insight, University College Cork
Laurent Simon	LaBRI, Bordeaux Institute of Technology
Sebastien Tabary	CRIL, Université d'Artois
Cyril Terrioux	LSIS, Université d'Aix-Marseille
Elise Vareilles	Mines Albi, Université de Toulouse
Nadarajen Veerapen	University of Stirling
Philippe Vismara	LIRMM, SupAgro
Mohamed Wahbi	Insight, University College Cork
Bruno Zanuttini	GREYC, Université de Caen Normandie
Matthias Zytnicki	INRA, Université de Toulouse

Rapporteurs Additionnels

Arcangioli, Robin; Baud-Berthier, Guillaume; Maamar, Mehdi; Armant, Vincent; Briot, Nicolas; Perez, Guillaume; Barreau, Kevin; Le Garrec, Marion; Ziat, Ghiles.

Préface

Les Journées Francophones de Programmation par Contraintes représentent chaque année une précieuse opportunité de rencontre, favorisant les échanges et la dissémination de résultats. L'édition 2016 m'a offert un poste d'observation privilégié sur les activités de la communauté, poste duquel la vue est imprenable et exaltante.

Tout d'abord, j'ai été impressionné par la qualité des articles soumis. Le programme scientifique offert aux participants est remarquable, et il démontre l'excellence de la "francophonie" dans ce domaine de recherche. Tout n'est pas parfait. Le nombre d'articles soumis (34, contre 49, 46 et 44 lors des trois dernières éditions) est le plus bas depuis de nombreuses années. Je ne m'aventurerai pas à y voir une tendance, pas plus que je ne proposerai d'explication, mais ce n'est certainement pas une bonne nouvelle.

Ensuite, j'ai vraiment été enchanté et admiratif du travail de relecture des membres du comité de programme. Le comité rassemble 34 chercheurs de 7 pays, 19 villes et 24 institutions.

Les JFPC représentent avant tout une opportunité d'échange, et n'ont donc pas vocation à être sélectives. Le taux d'acceptation de cette édition (97%!) en témoigne. Mais loin de faire baisser la qualité des rapports, ce fait a au contraire incité beaucoup de rapporteurs à fournir une quantité considérable de commentaires très constructifs. L'ensemble du comité a en outre consciencieusement et démocratiquement choisi les trois articles étudiants finalistes pour le prix AFIA de la meilleure contribution, présentés lors d'une session spéciale. Plus d'un tiers des articles soumis avaient un étudiant comme auteur principal, et chacun des finalistes peut être extrêmement fier étant donné la qualité des articles soumis.

J'ai également pu mesurer la somme d'efforts déployés par le comité d'organisation pour proposer un congrès qui sera, j'en suis absolument certain, une grande réussite. Ces journées lanceront à Montpellier un véritable "tour de France de la programmation par contraintes" qui fera ensuite étape à Bordeaux avec SAT 2016 pour se conclure à Toulouse pour CP 2016. Je souhaite au plus grand nombre de participer aux trois!

Enfin, un alléchant et éclectique programme de présentations invitées viendront parachever le travail des auteurs, rapporteurs et organisateurs pour rendre ce congrès agréable autant qu'enrichissant :

- Malik Ghallab, directeur de recherche au CNRS et directeur du LAAS de 2003 à 2006, présentera ses travaux récents sur des approches CSP pour la planification de tâches sous incertitude en Robotique.
- Benoit Rottembourg, directeur associé Pricing & Revenue Management à Eurodecision, présentera les problèmes induits par les sytèmes de "pricing" qui représentent un enjeux économique majeur.
- Johan Thapper, maître de conférences à l'université Paris-Est, présentera ses résultats récents, en particulier une caractérisation complète de la complexité des classes de CSP pondérés définies par des langages de contraintes.

C'est donc avec une grande fierté de faire partie de cette communauté et avec un optimisme renouvelé que je m'apprête à passer le témoin au prochain président du comité de programme.

Emmanuel Hebrard
Président du comité de programme des JFPC 2016

Table of Contents

Contingence, Contrôlabilité et Observabilité en Planification Temporelle

Malik Ghallab

LAAS-CNRS, Université de Toulouse, France

malik.ghallab@laas.fr

Résumé

La planification de tâches requiert souvent une représentation explicite du temps. On souhaite une représentation suffisamment flexible pour permettre de différer à la phase d'exécution l'instantiation des variables temporelles du plan synthétisé. Dans ce sens, les approches du type CSP permettent de maintenir, en cours de planification comme à l'exécution, un ensemble de solutions consistantes, sans se restreindre à une seule qu'il serait difficile de réaliser.

On se heurte alors à la difficulté classique des variables contingentes, qui correspondent à des événements hors du contrôle de l'agent, ou des durées qu'il ne maîtrise pas. Ces variables aléatoires ne peuvent être fixées ou contraintes par la planification. Le plan synthétisé doit cependant rester controllable : l'agent doit pouvoir à tout moment choisir les variables contrôlables, ayant observé les événements contingents qui se sont produits avant (*contrôlabilité dynamique*).

De nombreux travaux ont permis très récemment de maîtriser l'algorithmique de la contrôlabilité dynamique pour des représentations relativement simples (e.g., STN). Ces travaux supposent que tous les événements contingents qui influencent l'activité de l'agent sont observables. Ce n'est pas toujours la cas.

Comment vérifier que le plan est dynamiquement contrôlable si tout n'est pas observable ? Si le plan n'est pas contrôlable, que faut-il observer pour qu'il le soit ? Comment intégrer à la planification l'ajout des actions nécessaires à la contrôlabilité ?

L'exposé proposera un tour d'horizon assez large sur ces questions en les illustrant, sans rentrer dans les détails techniques, par des travaux récents, en particulier ceux du LAAS autour du système de planification FAPE.

Références

[1] M. Ghallab, D. Nau and P. Traverso. Automated Planning and Acting. Cambridge University Press, (in press, http ://www.laas.fr/planning), 2016

[2] A. Bit-Monnot, M. Ghallab, F. Ingrand. Which Contingent Events to Observe for the Dynamic Controllability of a Plan. Proc. IJCAI, 2016

[3] F. Dvořák, R. Barták, A. Bit-Monnot, F. Ingrand, M. Ghallab. Planning and Acting with Temporal and Hierarchical Decomposition Models. Proc. ICTAI, 115-121, 2014

50 nuances de prix : entre élasticité et contraintes

Benoit Rottembourg

Eurodecision, Versailles, France
LINA - Laboratoire d'Informatique de Nantes Atlantique

benoit.rottembourg@eurodecision.com

Résumé

Depuis 10 ans, les français ont massivement adopté internet comme boutique pour faire leurs courses. Le web représente aujourd'hui 700 millions d'actes d'achat par an, pour un montant proche de 60 milliards d'euros, soit 9% de nos achats (hors alimentation).

Derrière chaque transaction, il y a un choix de l'internaute. Un choix basé sur une envie, bien souvent une image et ... un prix. Il n'est pas rare qu'en amont d'un acte d'achat réussi - ou raté - nous ayons visualisé, plus ou moins consciemment, plus d'une centaine de prix différents, pour peu ou prou le même produit. De toute évidence, ces prix bougent. Le prix d'un hôtel à Paris bouge par exemple plusieurs fois par semaine, parfois par jour. Ils bougent de plus en plus fréquemment, et ipso facto, de plus en plus «automatiquement». Des milliers d'analystes pricing (en l'occurrence des revenue managers dans l'hôtellerie), servis par des logiciels puissants, ou des feuilles excel replètes sont désormais en charge de fixer ces prix. Leurs objectifs s'expriment le long de plusieurs axes : maximiser une marge, prendre une part de marché, augmenter un chiffre d'affaires.

L'objet de cette présentation est d'illustrer quelques-uns des principaux mécanismes qui expliquent la difficulté de la fixation des prix. Nous insisterons sur le triptyque «Elasticité du comportement client/ Contraintes marketing / Maximisation de la marge» qui conditionne les algorithmes de pricing des sites marchands et des comparateurs de prix. En nous inspirant des univers du transport, du tourisme et de la grande distribution, nous chercherons à décrire la richesse des contraintes en jeu ainsi que la volumétrie des problèmes algorithmiques à résoudre. Nous ne pourrons nous empêcher de décrire quelques méthodes de résolution couramment utilisées pour cette famille de problèmes d'optimisation combinatoire stochastique avec recours. Nous insisterons sur la dynamique du processus de fixation des prix qui contraint fortement la recherche de solutions. Nous conclurons par un retour d'expérience nuancé par la forte sensibilité de ces algorithmes à la qualité de la donnée et donc à la connaissance des élasticités évoquées.

Références

[1] T Benoist, E Bourreau, Y Caseau, B Rottembourg. Towards Stochastic Constraint Programming : A Study of Onine Multi-Choice Knapsack with Deadlines. In *Proceedings : Principles and Practice of Constraint Programming — CP 2001*, pages 61–76, Paphos, Cyprus, 2001

[2] K. Talluri, G van Ryzin. The Theory and Practice of Revenue Management. *Springer*, 2004

[3] M. Ferguson, T. Bodea. Pricing Segmentation and Analytics. *Business Expert Press*, 2012

Cohérences et dichotomies parmi des CSP pondérés

Johan Thapper

Université Paris-Est, LIGM (UMR 8049), Marne-la-Vallée, France

johan.thapper@u-pem.fr

Résumé

Nous considérons des CSP pondérés dans un cadre de restrictions de contraintes pondérées. Plus précisément, nous fixons un ensemble de fonctions de coût, appelé le langage de contraintes, et on considère le problème de minimiser une somme de contraintes pondérées où les fonctions de coût viennent du langage. On se restreint à l'étude des fonctions de coût qui prennent des valeurs rationnelles ou l'infini positif. En particulier, ce problème permet de modéliser deux types de problème d'optimisation classiques sur des instances de CSP :

— Min CSP où l'objectif est de minimiser le nombre de contraintes non satisfaites ;

— Min Sol où l'objectif est de minimiser une fonction objectif sur l'ensemble d'affectations satisfaisantes.

On s'intéresse à la question suivante : pour quels langages, le problème peut-il être résolu en temps polynomial ?

Les notions de cohérence en général et la cohérence d'arc en particulier sont des outils de base pour les solveurs et pour l'étude des classes polynomiales. Notre étude démarre avec la notion de cohérence d'arc d'un réseau de contraintes classique. Cette notion et ses généralisations nous amènent ensuite aux notions correspondantes des CSP pondérés, où il s'avère que ces notions donnent une réponse presque exhaustive à notre question ci-dessus.

Dalmau et Pearson [3] caractérisent les CSP de largeur 1, c'est-à-dire les CSP pour lesquels il suffit d'établir la cohérence d'arc une seule fois pour les résoudre, sans avoir besoin de backtracking. La classe de CSP de largeur 1 contient par exemple les CSP ayant un langage de contraintes max-closed. Plus tôt, Feder et Vardi [4] avaient déjà considéré cette notion dans un contexte de Datalog. Dans ce contexte, la notion se généralise naturellement à la notion de largeur relationnelle k, avec une notion de cohérence liée à chaque niveau k. Ils ont conjecturé que les CSP de largeur bornée (de largeur relationnelle k pour un k fixe) sont ceux qui n'ont pas la "capacité à compter". Récemment, cette conjecture a été résolue, culminant dans le travail de Barto et Kozik [1]. D'une certaine manière, alors, on comprend maintenant quels CSP classiques peuvent être résolus en établissant une cohérence locale.

Suivons alors le sentier correspondant pour le CSP pondéré. De nombreuses notions de cohérence d'arc pondéré ont été proposées. Cooper, de Givry et Schiex [2] introduisent la cohérence OSAC (Optimal Soft Arc Consistency) qui peut être établie en résolvant un programme linéaire (LP). Le problème dual de ce programme peut également être obtenu comme une relaxation du CSP pondéré. Cette relaxation s'appelle la BLP (Basic Linear Programming relaxation). Thapper et Živný [6] caractérisent les CSP pondérés pour lesquels l'optimum coïncide avec l'optimum de la BLP. Donc, pour ces CSP pondérés, il suffit de résoudre la BLP une seule fois, sans avoir besoin de backtracking.

Le papier de Feder et Vardi est plus connu pour poser la question suivante : le CSP restreint par un langage arbitraire (mais fixe) de relations, est-il toujours soit résoluble en temps polynomial, soit NP-dur ? Cette question, sous le nom de la conjecture de dichotomie, est toujours ouverte. Par contre, la question correspondante pour les Min CSP (et plus généralement pour la classe de CSP pondérés avec des fonctions de coût prenant des valeurs rationnelles finies), a été résolue. Thapper et Živný [7] démontrent que, soit l'optimum du Min CSP coïncide avec l'optimum de la BLP, soit le Min CSP est NP-dur. Plus récemment, Kolmogorov, Krokhin et Rolínek [5] ont donné une dichotomie conditionnelle pour tous les CSP pondérés : si la conjecture de Feder et Vardi est vraie, alors le

CSP pondéré restreint par un langage de contraintes est soit résoluble en temps polynomial, soit NP-dur.

La BLP se positionne à l'extrémité d'une hiérarchie de relaxations LP s'appelant la hiérarchie de Sherali-Adams. Les niveaux supérieurs de cette hiérarchie correspondent aux niveaux supérieurs de largeur relationnelle des CSP classiques. En utilisant la caractérisation de largeur bornée, Thapper et Živný [8] donnent une caractérisation pour les CSP pondérés qui sont résolubles en utilisant les "cohérences" dérivant de cette hiérarchie. Cette caractérisation implique en outre une dichotomie des problèmes de type Min Sol (avec une restriction légère sur la fonction objectif) : soit un problème Min Sol est résoluble par le troisième niveau de la hiérarchie de Sherali-Adams, soit le problème est NP-dur.

Références

[1] Libor Barto and Marcin Kozik. Constraint Satisfaction Problems Solvable by Local Consistency Methods. *Journal of the ACM*, 61(1), Article No. 3, 2014. (An extended abstract appeared in FOCS 2009).

[2] Martin C. Cooper, Simon de Givry, and Thomas Schiex. Optimal soft arc consistency. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 68–73, 2007.

[3] Víctor Dalmau and Justin Pearson. Closure functions and width 1 problems. In *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP99)*, pages 159–173, 1999.

[4] Tomás Feder and Moshe Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction : A Study through Datalog and Group Theory. *SIAM Journal on Computing*, 28(1) :57–104, 1998. (An extended abstract appeared in STOC 1993).

[5] Vladimir Kolmogorov, Andrei A. Krokhin, and Michal Rolínek. The complexity of general-valued CSPs. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2015)*, 2015.

[6] Johan Thapper and Stanislav Živný. The power of linear programming for valued CSPs. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, pages 669–678, 2012.

[7] Johan Thapper and Stanislav Živný. The complexity of finite-valued CSPs. In *Proceedings of the 45th ACM Symposium on the Theory of Computing (STOC 2013)*, pages 695–704, 2013.

[8] Johan Thapper and Stanislav Živný. Sherali-Adams relaxations for valued CSPs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP 2015)*, pages 1058–1069, 2015.

Programmation par contraintes stochastiques pour le General Game Playing avec informations incomplètes

Frédéric Koriche Sylvain Lagrue Eric Piette *Sébastien Tabary

Université Lille-Nord de France CRIL - CNRS UMR 8188 Artois, F-62307 Lens
{koriche,lagrue,epiette,tabary}@cril.fr

Résumé

Le *game description language* avec informations incomplètes (GDL-II) est assez expressif pour représenter les jeux stochastiques multi-agents avec observation partielle. Malheureusement, une telle expressivité n'est pas possible sans un prix : le problème consistant à trouver une stratégie gagnante est $\mathsf{NExp}^{\mathsf{NP}}$-hard, une classe de complexité qui est bien au-delà de la portée des *solvers* actuels. Dans ce papier, nous identifions un fragment Pspace-complete de GDL-II, où les agents partagent les mêmes observations (partielles). Nous montrons que ce fragment peut être encapsulé dans un problème de satisfaction de contraintes stochastiques décomposable (SCSP) qui, par tour, peut être résolu en utilisant des techniques de programmation par contraintes usuelles. Dès lors, nous avons développé un algorithme de décisions séquentielles fondé sur les contraintes pour les jeux GDL-II exploitant la propagation par contraintes, l'évaluation Monte-Carlo et la détection de symétries. Notre algorithme, vérifié sur une large variété de jeux, surpasse aisément l'état de l'art des algorithmes du *general game playing* (GGP).

Abstract

The game description language with incomplete information (GDL-II) is expressive enough to capture partially observable stochastic multi-agent games. Unfortunately, such expressiveness does not come without a price : the problem of finding a winning strategy is $\mathsf{NExp}^{\mathsf{NP}}$-hard, a complexity class which is far beyond the reach of modern constraint solvers. In this paper we identify a Pspace-complete fragment of GDL-II, where agents share the same (partial) observations. We show that this fragment can be cast as a decomposable stochastic constraint satisfaction problem (SCSP) which, in turn, can be solved using general-purpose constraint programming techniques. Namely, we develop a constraint-based sequential decision algorithm for GDL-II games which exploits constraint propagation, Monte-Carlo sampling, and symmetry detection. Our algorithm, validated on a wide variety of games, significantly outperforms the state-of-the-art general game playing algorithms.

*Papier doctorant : Eric Piette est auteur principal.

1 Introduction

De toutes les activités humaines, les jeux amènent l'un des exemples de comportement intelligent les plus illustratifs. En effet, un joueur doit faire face à de nombreuses taches complexes, telles que la compréhension de règles abstraites, l'évaluation de la situation courante, choisir le meilleur mouvement possible et finalement élaborer une stratégie gagnante. Dans le cadre de l'intelligence artificielle, le challenge *General Game Playing* (GGP) [8, 7] propose de développer des joueurs informatisés qui comprennent les règles de jeux précédemment inconnus et d'apprendre à y jouer efficacement sans intervention humaine.

Dans le *General Game Playing*, les règles d'un jeu sont décrites dans un formalisme de représentation déclaratif de haut niveau, nommé le *Game Description Language* (GDL). La première version de ce langage (GDL-I) est restreinte aux jeux déterministes avec informations complètes [14]. Tandis qu'un jeu GDL-I peut impliquer plusieurs joueurs, chaque participant a une connaissance complète de l'état courant du jeu, les actions de ces adversaires et des effets déterministes de l'ensemble des actions.

Dans le but de soulever ces restrictions, Schiffel et Thielscher [17, 18] ont récemment proposé une nouvelle version du *Game Description Language* (GDL-II) permettant de représenter les jeux avec

informations incomplètes. Dans un jeu GDL-II, les joueurs peuvent avoir un accès limité aux informations de l'état courant et les effets de l'ensemble des actions sont incertains. A ce titre, GDL-II est assez expressif pour représenter les jeux stochastiques avec observation partielle (POSGs), qui couvrent une large variété de problèmes multi-agents de décisions séquentielles. Cependant, une telle expressivité est impossible sans payer le prix : le problème consistant à trouver une stratégie gagnante est NEXP^{NP}-hard [10], une classe de complexité qui est bien au-delà de la classe usuelle de complexité pour les jeux avec informations complètes (PSPACE). Bien que différents algorithmes ont été conçus pour résoudre des instances spécifiques de POSGs tel que le *Contract Bridge* [9] et *Poker* [2], ce sont tous des programmes dédiés qui reposent fortement sur la connaissance humaine des règles du jeu et sur des fonctions d'évaluations. Par contre, le développement d'un joueur de *General Game Playing* pour POSGs semble extrêmement difficile suite à la barrière de la complexité.

Malgré ce résultat théorique, plusieurs algorithmes GGP ont été récemment développés pour résoudre certains fragments de GDL-II. Parmi eux, les jeux déterministes mono agent avec informations incomplètes [5] et les jeux stochastiques multi-agents avec informations complètes au travers de notre algorithme MAC-UCB [13]. Cette dernière approche repose sur les techniques de programmation par contraintes qui, en pratique, se sont révélées gagnantes. Notamment au cours de la dernière compétition internationale GGP dénommée *The Tiltyard Open* 2015, nous avons fait participer MAC-UCB sous le nom de *WoodStock* (We Our Own Development STOChastic toolKit) qui a terminé 2$^{\text{ème}}$ au cours de la phase de qualification et 3$^{\text{ème}}$ en phase terminale [1]. Notons d'ailleurs que cette compétition est dédiée aux jeux déterministes en GDL-I dans lesquels notre algorithme MAC-UCB ne peut profiter de sa principale fonctionnalité de filtrage via les variables stochastiques.

Notre présente étude a pour objectif d'étendre MAC-UCB à la gamme des jeux GDL-II. La contribution principale de cet article est résumé ainsi : (i) nous présentons un fragment important de GDL-II, où les joueurs partagent les mêmes observations (partielles) et qui peut être représenté par un SCSP ; (ii) Nous proposons un algorithme de décisions séquentielles à base de contraintes pour GDL-II qui exploitent la propagation par contraintes, l'évaluation Monte-Carlo et la détection de symétries ; (iii) Nous apportons une étude expérimentale comparative sur de nombreux jeux GDL, incluant des jeux déterministes, des jeux stochastiques et des jeux stochastiques avec observations partielles (avec informations partagées).

Les résultats expérimentaux montrent que la technique de programmation par contraintes stochastiques conduite par les symétries surpasse aisément l'état de l'art des algorithmes de *General Game Playing*.

2 GGP avec informations incomplètes

Les problèmes étudiés en GGP sont des jeux séquentiels finis. Chaque jeu implique un nombre fini de joueurs et un nombre fini d'états, incluant un état initial distinct et un ou plusieurs états terminaux. A chaque tour de jeu, chaque joueur possède un nombre fini d'actions (nommé coups légaux) ; l'état courant du jeu est mis à jour par les conséquences simultanées de l'action de chaque joueur (qui peut être *noop* pour ne rien faire). Le jeu commence avec un état initial et après un nombre fini de tours se termine sur un état terminal au cours duquel un score compris entre 0 et 100 est attribué à chaque joueur. Dans un jeu stochastique, un joueur distinct, souvent faisant référence à la chance, choisit ses actions aléatoirement selon une distribution de probabilités définie sur ses coups légaux. Dans un jeu à informations incomplètes, certains aspects (nommé *fluents)* de l'état courant du jeu ne sont pas complètement révélés aux agents. Nous allons nous concentrer sur les jeux stochastiques à informations partagées dans lesquels à chaque tour l'ensemble des agents ont les mêmes (parfois incomplètes) informations sur l'état du jeu.

La syntaxe de GDL-II. GDL est un langage déclaratif permettant de représenter les jeux finis. Basiquement, un programme GDL est un ensemble de règles décrites en logique du premier ordre. Les joueurs et les objets du jeu (pièces, dés, lieux, cases, etc.) sont décrits par des constantes tandis que les fluents et les actions sont décrites par des termes du premier ordre. Les atomes d'un programme GDL sont construits selon un ensemble fini de symboles de relations et de symboles de variables. Quelques symboles ont un sens spécifique dans un programme et sont décrits dans le tableau 1. Par exemple, dans le *TicTacToe*, *legal(Alice,mark(X,Y))* indique que le joueur Alice à l'autorisation de jouer la case (X,Y) du plateau. En GDL-II, les deux derniers mots-clés du tableau sont ajoutés pour représenter les jeux stochastiques (*random*) et les jeux à informations partielles (*sees*).
Les règles d'un programme GDL sont des clauses de Horn du premier ordre. Par exemple, la règle :

$$\texttt{sees}(\texttt{Alice}, \texttt{cell}(\texttt{X}, \texttt{Y}, \texttt{o})) \leftarrow \texttt{does}(\texttt{Alice}, \texttt{mark}(\texttt{X}, \texttt{Y}))$$

1. L'ensemble des matchs sont consultables ici : www.ggp.org/view/tiltyard/matches/#tiltyard_open_20151204_2

Mots-clés	Description
role(P)	P est un joueur
init(F)	Le fluent F est présent à l'état initial
true(F)	Le fluent F est présent
legal(P, A)	Le joueur P peut réaliser l'action A
does(P, A)	Le joueur P réalise l'action A
next(F)	Le fluent F est présent dans le prochain état
terminal	L'état courant est terminal
goal(P, V)	Le joueur P obtient un score V
sees(P, F)	Le joueur P perçoit F
random	Le joueur "chance"

TABLE 1 – les mots-clés GDL-II

indique que Alice peut voir l'effet provoqué par le marquage des cases du plateau. Dans le but de représenter un jeu séquentiel fini, un programme GDL doit obéir à certaines conditions syntaxiques définies à l'aide de termes et de relations apparaissant dans les règles. Nous référons le lecteur à [14, 22] pour une analyse détaillée de ces conditions.

La sémantique de GDL-II. Pour un entier positif n, soit $[n] = \{1, \cdots, n\}$. Pour un ensemble fini S, soit Δ_S signifiant la probabilité sur S, c'est à dire, l'ensemble de toutes les distributions de probabilités sur S. De nombreuses descriptions de jeux à informations incomplètes sont proposées dans la litérature (voir e.g. [17]). Nous nous concentrons ici sur une variante de [5].

Formellement, un jeu stochastique avec informations partielles avec des coups légaux (POSG) est un tuple $G = \langle k, S, s_0, S_g, A, L, P, B, R\rangle$, où $k \in \mathbb{N}$ est le nombre de joueurs, S est l'ensemble fini des états, incluant l'état initial s_0, et un sous ensemble S_g d'états terminaux. A est un ensemble fini d'*actions*. Comme d'habitude le *profil d'actions* est un tuple $\boldsymbol{a} = \langle a_1, \ldots, a_k\rangle \in A^k$; par a_p, nous définissons l'action du joueur p, et par $\boldsymbol{a}_{-p}$ le profil d'actions $\langle a_1, \ldots, a_{p-1}, a_{p+1}, \ldots, a_k\rangle$ des joueurs restants. $L : [k] \times S \to 2^A$ définit l'ensemble des *actions légales* $L_p(s)$ du joueur p à l'état s ; nous supposons $L_p(s) = \emptyset$ pour tous $s \in S_g$. $P : S \times A^k \hookrightarrow \Delta_S$ est la *fonction de transition partielle de probabilité*, qui attribue à chaque état $s \in S$ et chaque profil d'actions $\langle a_1, \ldots, a_k\rangle \in L(s)$ à une distribution de probabilités sur S. $B : [k] \times S \to \Delta_S$ est la *fonction de croyance* qui attribue chaque joueur $p \in [k]$ et chaque état $s \in S$ à une distribution de probabilités $B_p(s)$ sur S, représentant l'état de croyance de p à s. Finallement, $R : [k] \times S_g \to [0, 1]$ est la *fonction de score* qui attribue chaque joueur $p \in [k]$ et chaque état terminal $s \in S_g$ à une valeur $R_p(s) \in [0, 1]$, représentant le score de p sur s.

Avec ces notions en tête, le POSG G associe au programme GDL-II G défini comme suit. Soit H dénotant la base Herbrand (i.e. l'ensemble des termes) de G ; A (resp. F) est l'ensemble des termes d'actions (resp. fluent) apparaissant dans H. Nous utilisons $\mathtt{G} \models \mathtt{A}$ pour indiquer que l'atome A est vrai dans l'ensemble de réponses uniques de G. Le nombre k de termes p tel que $\mathtt{role(p)} \in \mathtt{G}$, indique l'ensemble des joueurs.

Chaque état s est un sous ensemble de F. Particulièrement, l'état initial s_0 est $\{f : \mathtt{G} \models \mathtt{init(f)}\}$, et tout état terminal est l'ensemble de fluents $s = \{f_1, \cdots, f_n\}$ tel que $\mathtt{G} \cup s^{\mathtt{true}} \models \mathtt{terminal}$, où $s^{\mathtt{true}}$ est l'ensemble de faits $\{\mathtt{true(f_1)}, \cdots, \mathtt{true(f_n)}\}$. L'ensemble $L_p(s)$ des actions légales pour le joueur p à l'état s est donné par $\mathtt{G} \cup s^{\mathtt{true}} \models \mathtt{legal(p, a)}$. Spécialement, $L_0(s)$ indique l'ensemble des actions légales pour le joueur "chance" (random). Tout profil d'actions (étendu au joueur "chance") $\boldsymbol{a} = \langle a_0, a_1, \ldots, a_k\rangle \in L_0(s) \times L_1(s) \times \cdots \times L_k(s)$ indique un successeur s' de s donné par $\{f : \mathtt{G} \cup s^{\mathtt{true}} \cup \boldsymbol{a}^{\mathtt{does}} \models \mathtt{next(f)}\}$, où $\boldsymbol{a}^{\mathtt{does}}$ est l'ensemble de faits $\{\mathtt{does(a_0)}, \mathtt{does(a_1)}, \ldots, \mathtt{does(a_k)}\}$. La distribution de probabilités $P(s, \boldsymbol{a}_{-0})$ sur tous ces successeurs est une distribution uniforme, i.e. $P(s, \boldsymbol{a}_{-0})(s') = 1/|L_0(s)|$. L'état de croyance $B_p(s)$ du joueur p de tout successeur s' de s est donné par la distribution jointe $\prod_{f \in F} P(f)$, où $P(f) = 1$ si $\mathtt{G} \cup s^{\mathtt{true}} \cup \boldsymbol{a}^{\mathtt{does}} \models \mathtt{sees(p, f)}$, et $P(f) = 1/2$ sinon. Finallement, le score $R_p(s)$ du joueur p à l'état terminal s est la valeur v tel que $\mathtt{G} \cup s^{\mathtt{true}} \models \mathtt{goal(p, v)}$.

Un fragment Pspace de GDL-II. Un jeu avec informations partagées représente tout jeu stochastique avec informations partielles G dans lequel à chaque tour tous les joueurs partagent le même état de croyance, i.e. $B_1(s) = \cdots = B_k(s)$ pour tout état $s \in S$. Nous utilisons ici $B(s)$ pour indiquer l'état de croyance commun dans s. Clairement, tout jeu avec informations partagées peut être converti dans un jeu stochastique complètement observable, en remplaçant la fonction de transition P et la fonction de croyance B par une nouvelle fonction de croyance $Q : S \times A^k \hookrightarrow \Delta_S$ définie ainsi :

$$Q(s, \boldsymbol{a})(s') = \prod_{t \in S} P(s, \boldsymbol{a})(t) \cdot B(t)$$

pour tout $\boldsymbol{a} \in L(s)$. En d'autres mots, $Q(s, \boldsymbol{a})(s')$ est la probabilité d'observer s' après avoir effectué le profil d'actions $\boldsymbol{a}$ à l'état s. Etant donné que pour tout jeu G stochastique avec informations partagées, une politique jointe de G est une correspondance $\boldsymbol{\pi} : S \to A^k$, où $\pi_p(s)$ est la politique du joueur p, et $\boldsymbol{\pi}_{-p}(s)$ est la politique jointe des autres joueurs. Soit un vecteur de seuil $\boldsymbol{\theta} \in [0, 1]^k$, nous pouvons dire que $\boldsymbol{\pi}$ est une *politique gagnante* pour le joueur p si le score espéré de p par rapport à $\boldsymbol{\pi}$ est plus grand que θ_p.

Basé sur la notion d'informations partagées, nous pouvons maintenant examiner quelques restrictions des programmes GDL-II qui garantissent ensemble que la recherche d'une politique gagnante est dans

PSPACE. Notamment, un programme GDL-II $\mathtt{G}$ a une profondeur limitée si le nombre de termes dans l'univers d'Herbrand de $\mathtt{G}$ est polynomial sur $|\mathtt{G}|$. Si $\mathtt{G}$ est limité et que chaque règle de $\mathtt{G}$ a un nombre constant de variables, alors $\mathtt{G}$ est *propositionel*. Pour un entier T, $\mathtt{G}$ est l'*horizon* T si tout état terminal est atteignable après au plus T rounds. Pour finir, $\mathtt{G}$ est à informations partagées si pour chaque joueur p, chaque fluent f, chaque état s, et chaque profil d'actions $\boldsymbol{a}$, on a si $\mathtt{G} \cup s^{\mathtt{true}} \cup \boldsymbol{a}^{\mathtt{does}} \models \mathtt{sees}(\mathtt{p}, \mathtt{f})$, alors $\mathtt{G} \cup s^{\mathtt{true}} \cup \boldsymbol{a}^{\mathtt{does}} \models \mathtt{sees}(\mathtt{q}, \mathtt{f})$, pour tout joueur $q \in [k]$.

Théorème 1. *Soit $\mathcal{G}_T \subseteq$ GDL-II le fragment propositionel, des programmes d'informations partagées d'horizon T. Alors, la recherche consistant à trouver une politique gagnante de $\mathcal{G}_T$ est* PSPACE*-complète.*

Preuve (Schéma) Soit $\mathcal{G}_T$ inclut les jeux stochastiques à informations complètes comme un cas particulier, le problème est PSPACE-hard. Pour tout jeu fini et à profondeur limitée $\mathtt{G} \in \mathcal{G}_T$, une politique gagnante peut être trouvée en temps $f(|\mathtt{G}|)$ et $g(|\mathtt{G}|)$ espaces utilisant les machines de Turing alternativement stochastiques, i.e. une machine de Turing qui inclut des états stochastiques (pour `random`), des états existentiels (pour le joueur p), et des états universels (pour tous les autres joueurs). Soit $\mathtt{G}$ propositionel, le nombre de fluents et le nombre d'actions sont polynomiaux sur $|G|$, qui ensemble impliquent que $g(|\mathtt{G}|)$ est polynomial. A chaque état de jeu, la machine de Turing alternativement stochastique peut deviner un profil d'actions en utilisant ces états existentiels. Soit $k \leq |\mathtt{G}|$, le prochain état de jeu pouvant être trouvé en utilisant un nombre polynomial d'états universels et d'états stochastiques. Ceci, avec le fait que la machine de Turing peut trouver un état terminal avec au plus T tours, implique que $f(|\mathtt{G}|)$ est aussi polynomial. Pour finir, soit toutes machines de Turing alternativement stochastiques utilisant un temps et un espace polynomial peut être simulé par NPSPACE (see e.g. [1]) donc, en utilisant le théorème de Savitch NPSPACE = PSPACE, il s'en suit que $\mathcal{G}_T$ est dans PSPACE, ce qui confirme le théorème. □

3 Le cadre SCSP

Empruntant la terminologie de [23], les réseaux de contraintes stochastiques étendent le cadre standard CSP en introduisant des variables stochastiques en plus des variables de décisions usuelles. Nous nous concentrons ici sur une généralisation du modèle original SCSP qui représente de multiples agents et les distributions de probabilités conditionnelles sur les variables stochastiques.

Définition 1. Un problème k-joueur de *Satisfaction de Contraintes stochastiques (SCSP)* est un 6-tuple $N = \langle V, Y, D, C, P, \theta \rangle$, tel que $V = (v_1, \cdots, v_n)$ est un tuple fini de variables, $Y \subseteq V$ est l'ensemble des *variables stochastiques*, D est une correspondance de V sur les *domaines*, C est un ensemble de *contraintes*, P est un ensemble de *tables de probabilités conditionnelles*, et $\theta \in [0,1]^k$ est un *seuil*.

- Chaque contrainte dans C est une paire $c = (scp_c, val_c)$, tel que scp_c est un sous ensemble de V, nommé la *portée* de c, et val_c est une correspondance de $D(scp_c)$ vers $([0,1] \cup \{-\infty\})^k$.
- chaque table de probabilités conditionnelles dans P est un triplet $(y, scp_y, prob_y)$, où $y \in Y$ est une variable stochastique, scp_y est un sous ensemble de variables apparaissant avant y dans V, et $prob_y$ est une correspondance de $D(scp_y)$ vers une distribution de probabilité $D(y)$.

Par X, nous indiquons l'ensemble $V \setminus Y$ de *variables de décisions*. Quand $y \in Y$ est une variable stochastique et $\tau \in D(scp_y)$ est un tuple de valeur dans la table de probabilités conditionnelles y, alors nous utilisons $P(y \mid \tau)$ qui indique la distribution $prob_y(\tau)$. En particulier, si $d \in D(y)$, alors $P(y = d \mid \tau)$ est la probabilité que y prend la valeur d donnée par τ.

Soit un sous ensemble $U = (v_1, \cdots, v_m) \subseteq V$, une *instanciation* de U est un assignement I de valeurs $d_1 \in D(v_1), \cdots, d_m \in D(v_m)$ aux variables $v_1, \cdots, v_m$, aussi écrites $I = \{(v_1, d_1), \ldots, (v_m, d_m)\}$. Une instantiation I sur U est *complète* si $U = V$. Soit un sous ensemble $U' \subseteq U$, nous utilisons $I_{|U'}$ qui indique la restriction de I vers U', c'est à dire, $I_{|U'} = \{(v_i, d_i) \in I : v_i \in U'\}$. La *probabilité* de I est donnée par :

$$P(I) = \prod_{y \in Y : scp_y \subseteq U} P(y = I_{|y} \mid I_{|scp_y})$$

Corrélativement, l'*utilité* d'une instanciation I sur U est donnée par

$$val(I) = \sum_{c \in C : scp_c \subseteq U} val(I_{|scp_c})$$

Notons que $val(I)$ est un tuple $(val_1(I), \cdots, val_k(I))$ assignant une utilité à chaque joueur. Une instanciation I est *localement cohérente* si $val_p(I) \neq -\infty$ pour chaque joueur p, c'est à dire, I satisfait chaque contrainte dure dans C. I est *globalement cohérente* (ou *cohérente*) si il peut être étendu à une instanciation complète I' qui est localement cohérente.

Une *politique* π pour le réseau N est un arbre où chaque noeud interne est étiqueté par une variable v et chaque arête est étiquetée par une valeur dans $D(v)$. Spécifiquement, les noeuds sont étiquetés selon l'ordre V : le noeud racine est étiqueté par v_1,

et chaque fils d'un noeud v_i est étiqueté par v_{i+1}. Les noeuds de décisions x_i ont un unique fils, et les noeuds stochastiques y_i ont $|D(y_i)|$ enfants. Enfin, chaque feuille dans π est étiquetée par l'utilité $val(I)$, où I est l'instanciation complète spécifiée par le chemin de la racine de π à chaque feuille. Soit $\mathcal{L}(\pi)$ l'ensemble de tous les instanciations complètes induites par π. Alors, l'*utilité attendue* de π est la somme de sa feuille d'utilité pondérée par leur probabilité. Formellement, $val(\pi) = \sum_{I \in \mathcal{L}(\pi)} P(I) val(I)$.

Une *politique* π est *faisable* si et seulement si $val(\pi) \geq (0, \ldots, 0)$, i.e. tout chemin dans π est globalement cohérent. Finalement, π est une *solution* de N si son utilité attendue est plus grande ou égale que le seuil $\theta = (\theta_1, \ldots, \theta_k)$. Cela implique que $val_p(\pi) \geq \theta_p$ pour tout joueur p.

Un *(decision) niveau* dans un SCSP est un tuple de variables $\langle X_i, Y_i \rangle$, où X_i est un sous ensemble de variables de décisions, Y_i est un sous ensemble de variables stochastiques, et de variables de décisions survenant avant toute variable stochastique [11]. Par extension :

Définition 2. Un *T-niveau k-joueur SCSP* est un k-player SCSP $N = \langle V, Y, D, C, P, \theta \rangle$, dans lequel V peut être partitionné en T *niveaux*, i.e. $V = (\langle X_1, Y_1 \rangle, \ldots, \langle X_T, Y_T \rangle)$, où $\{X_i\}_{i=1}^T$ est une partition de X, $\{Y_i\}_{i=1}^T$ est une partition de Y, et $scp_{y_i} \subseteq X_i$ pour chaque $i \in \{1, \ldots, T\}$ et chaque $y_i \in Y_i$. Si $T = 1$, N est appelé un *1-niveau* SCSP, et est défini par μSCSP.

Notons que la recherche d'une politique gagnante dans un SCSP est PSPACE-complète. Le problème reste dans PSPACE pour T-niveau k-joueurs SCSPs, car chaque nievau du problème est dans $\mathrm{NP}^{\mathrm{PP}}$.

SCSP pour GDL-II. En se basant sur ce cadre, la traduction d'un programme k-joueur $\mathtt{G} \in \mathcal{G}_T$ vers un T-niveau k-joueur SCSP est spécifié comme suit. Nous commençons par convertir G vers un ensemble de règles closes $\mathtt{G}'$, dont leur taille est polynomial dans $|\mathtt{G}|$ car G est propositionel. Pour $\mathtt{G}'$ nous associons un 1-niveau SCSP N avec $V = \langle g_t, \{f_t\}, \{a_t\}, y_t, \{f_{t+1}\} \rangle$, où g_t est une variable booléenne indiquant si le jeu a atteint un état terminal ; $\{f_t\}$ et $\{f_{t+1}\}$ sont les ensembles de variables stochastiques décrivant l'état du jeu au tour t et $t+1$, respectivement ; $y_t = a_{t,0}$ est une variable stochastique décrivant l'ensemble des coups légaux du joueur "chance" ; et $\{a_t\} = \{a_{t,1}, \ldots, a_{t,k}\}$ est l'ensemble des variables de décisions, chaque $a_{t,p}$ décrit l'ensemble des coups légaux du joueur p. A proprement parler, N n'est pas un 1-niveau SCSP, en raison des variables stochastiques $\{f_t\}$, mais le domaine de ces variables est un singleton, indiquant l'état courant du jeu. Ainsi, lors du processus de résolution, $\{f_t\}$ peut être supprimé par N pour produire un μSCSP.

Les clauses de Horn d'un programme GDL G peuvent être naturellement partitionnées en règles `init` décrivant l'état initial, en règles `legal` spécifiant les coups légaux de l'état courant, en règles `next` indiquant les effets des actions, en règles `sees` décrivant les observations de chaque joueur sur le jeu, et en règles `goal` définissant les scores des joueurs à l'état terminal. Les règles `init`, `legal` et `next` sont encodées en contraintes dures dans le réseau N. Les règles `sees` sont utilisées pour exprimer la table de probabilités conditionnelles de chaque variable stochastique f_{t+1}. La dernière variable stochastique y_t est associée à la distribution de probabilité uniforme sur les coups légaux du joueur "chance". La contrainte relation est extraite de la même manière que les domaines des variables, par identification de toutes les combinaisons autorisées des constantes. Similairement, les règles `goal` sont encodées par une contrainte souple sur N ; basé sur leurs sémantiques, les scores des joueurs sont toujours à 0 pour les états non terminaux et une valeur de $[0, 1]$ pour tout état terminal.

En répétant T fois ce processus de conversion, nous obtenons alors un T-niveau SCSP encodant le jeu G à T-horizon. Le seuil θ peut être ajusté selon la stratégie désidée : commençant par la valeur $\boldsymbol{\theta} = (0, \ldots, 0)$ qui autorise toutes les politiques faisables, on peut utiliser la valeur attendue comme une solution du SCSP courant comme un nouveau seuil, qui détermine un SCSP plus contraint défini sur les mêmes contraintes.

4 MAC-UCB-SYM

Sur la base du fragment de SCSP pour les jeux GDL, nous présentons maintenant notre technique de résolution dénommée MAC-UCB-SYM, une extension de l'algorithme MAC-UCB [13] qui exploite la détection de symétrie. Comme indiqué ci-dessus, le réseau de contraintes stochastiques d'un programme GDL est une séquence de μSCSPs, chacun associant au tour du jeu t dans $\{1, \ldots, T\}$. Pour chaque $\mu\mathrm{SCSP}_t$ dans $\{1, \ldots, T\}$, MAC-UCB recherche l'ensemble des politiques faisables en divisant le problème en 2 parties : un CSP et un μSCSP (plus petit que l'original). La première partie est résolue avec l'algorithme MAC et la seconde partie via l'algorithme FC dédié au SCSP. Puis, un échantillonnage avec borne de confiance est réalisé pour estimer l'utilité attendue pour chaque solution réalisable d'un $\mu\mathrm{SCSP}_t$. Les symétries sont exploitées pour éviter l'exploration inutile de μSCSPs : à chaque μSCSP résolu, les μSCSPs symétriques sont calculés et stockés avec leurs utilités attendues.

Technique de filtrage utilisant les symétries. Rappelons que la décision séquentielle associée à un jeu de stratégie est un problème d'optimisation. Classiquement, ce problème est abordé en recherchant une solution à une séquence de problèmes de satisfaction stochastiques. En pratique, à chaque tour notre joueur doit jouer, un laps de temps est dédié au choix de la prochaine action. Pendant ce temps, nous résolvons autant de μSCSPs que possible, c'est pourquoi éviter de résoudre des μSCSPs inutilement est un point clé. Comme les μSCSPs correspondent à une représentation des états du jeu avec les actions possibles de chaque joueur, il est possible de rencontrer des μSCSPs similaires qui correspondent à des états du jeu similaires. Par exemple, dans un *TicTactoe*, une fois que l'adversaire à marqué le centre, marqué l'un des quatres coins revient à une situation similaire. Dans notre approche, nous proposons de détecter et de stocker les μSCSPs symétriques dans le but de bénéficier de ceux déjà résolus.

Rappelons qu'une symétrie dans un ensemble D est une permutation sur D ou, de manière équivalente une bijection σ de D vers D. Couramment, de telles permutations peuvent être représentées comme un ensemble de *cycles*. Par exemple, si $D = \{1, 2, 3, 4\}$, alors la permutation σ spécifie que $\sigma(1) = 2, \sigma(2) = 3, \sigma(3) = 1$ et $\sigma(4) = 4$, peuvent aussi être décrites par $\{(1, 2, 3), (4)\}$, ou de manière encore plus compacte $\{(1, 2, 3)\}$ où les cycles unaires sont implicites. De nombreux types de symétries ont été proposés dans la litérature de la programmation par contraintes, notamment les *symétries de solutions* qui préservent l'ensemble des solutions(e.g. [12, 16]), et *constraint symmetries* qui préservent l'ensemble des contraintes [4].

MAC-UCB-SYM exploite les symétries de contraintes, caractérisées par les automorphismes de la microstructure complémentaire du réseau. Plus précisément, chaque μSCSP P au niveau t est spécifié selon la Section 3, est enrichi par une contrainte dure de portée $(\{f_t\}, \{a_t\}, y_t)$, où la relation associée est formée de tuples de la forme $(s, \boldsymbol{a})$, s définis sur $\{f_t\}$ et $\boldsymbol{a}$ le profil d'actions défini sur $\{a_t\}$ et y_t. Chaque tuple $(s, \boldsymbol{a})$ est vu comme un "nogood" si toute politique depuis s avec le profil d'actions $\boldsymbol{a}$ est prédite par UCB comme une "non-solution" (son utilité espérée est sous le seuil $\boldsymbol{\theta}$). Basé sur ce μSCSP P, nous calculons les automorphismes de la microstructure dérivés de P. La microstructure est un hypergraphe, qui peut aussi être décrit par un graphe biparti : la partie gauche étant l'ensemble des littéraux (paires de variable-valeur), la partie droite est l'ensemble des tuples incohérents de toutes les contraintes de μSCSP, et une arête correspond à l'occurence d'un littéral dans un tuple d'une contrainte. A partir de ce graphe, nous commençons par calculer les automorphimes et nous générons les μSCSPs symétriques. Puis, ces μSCSPs sont stockés avec leurs utilités attendues dans une base (en pratique une table de *hash*). Avant de résoudre un μSCSP donné, nous recherchons dans la base si le μSCSP correspond à un μSCSP symétrique déjà rencontré. Si c'est le cas, l'utilité associée est directement obtenu sans le résoudre.

Mise en pratique. Considérons un *TicTacToe* 2×2 contre un joueur jouant aléatoirement. Un joueur gagne si il remplit une ligne ou une colonne (les diagonales sont exclues). MAC-UCB-SYM joue avec le symbole "o". Supposons que le symbole "x" est déjà présent en $(1, 1)$ à l'état initial. La Figure 1 représente l'arbre de recherche obtenu par MAC-UCB-SYM, dans lequel une symétrie est apparue entre l'action `mark(1,2)` (le noeud de gauche en bleu) et l'action `mark(2,1)` (le noeud de droite en rouge). Après avoir résolu l'état initial μSCSP, MAC-UCB-SYM résout μSCSP_1^1 obtenu par l'action `mark(1,2)`. UCB renvoit une utilité attendue de 0.25 pour cet état. Après avoir calculé l'automorphime pour μSCSP_1^1, nous générons et stockons le μSCSP symétrique correspondant à la permutation $\{(cell, 1\_2\_o), (cell, 1\_2\_blank)\}$, $\{(cell, 2\_1\_blank), (cell, 2\_1\_o)\}$,$\{(action_o, 1\_2), (action_o, 2\_1)\}$. Puis MAC-UCB-SYM s'intéresse à μSCSP_1^2 : ici, la détection de symétrie ne révèle pas de nouveau automorphisme et UCB associe une utilité attendue de 0 pour ce μSCSP. Enfin le dernier μSCSP μSCSP_1^3 correspond à un μSCSP symétrique généré précédemment. Par conséquent, la même utilité 0.25 associée à μSCSP_1^1 est associée à μSCSP_1^3 sans lancer UCB.

MAC-UCB-SYM choisit une action parmi ceux avec l'utilité attendue la plus forte : ici, il sélectionne aléatoirement l'une des actions `mark(1,2)` ou `mark(2,1)`. Ainsi, l'idée est de jouer sur la même ligne ou la même colonne que le symbole "x" pour espérer un nul.

5 Résultats Expérimentaux

Dans ce cadre, nous présentons une série de résultats expérimentaux réalisés sur un *cluster* d'Intel Xeon E5-2643 CPU 3.3 GHz avec 64 GB de mémoire RAM et quatres coeurs sous Linux. Notre algorithme a été implémenté en C++ et nous utilisons NAUTY [15] pour la détection de symétrie.

Nous avons sélectionné 20 jeux déterministes décrits en GDL extraits de la base de jeux GDL du serveur Tiltyard [2], et 15 jeux stochastiques en GDL-II incluant 5 sans règles `sees`. La majorité des jeux GDL-II a été sélectionné du serveur Dresden [3]. Les expérimentations

2. http://tiltyard.ggp.org/
3. http://ggpserver.general-game-playing.de/

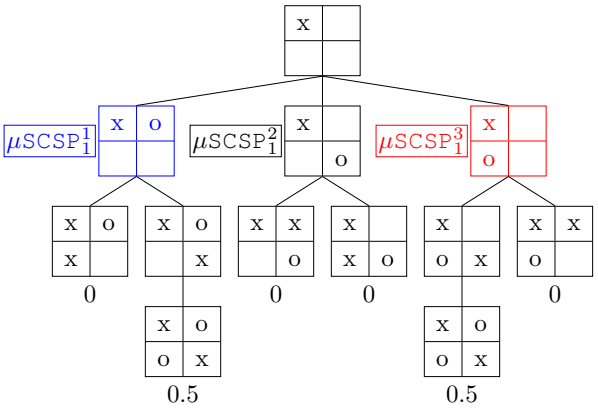


FIGURE 1 – L'arbre de recherche du *TicTacToe* 2 × 2 avec les scores obtenus par MAC-UCB-SYM pour chaque état terminal

ont été réalisées sur une grande variété de jeux pour un total de 40,000 matchs. Il est possible de retrouver plus en détail chaque jeu sur le site boardgamegeek.com.

Contexte. Les différentes compétitions de jeux sont organisées entre les différents joueurs de *General Game Playing*. Le premier joueur est une version multi-joueurs de l'algorithme UCT [21], qui représente l'état de l'art des joueurs GGP pour les jeux déterministes. Le second joueur est Sancho version 1.61c [4], un joueur basé sur la recherche Monte-Carlo dans les arbres de recherche élaboré par S. Draper and A. Rose, qui a gagné l'*International General Game Playing Competition* 2014 (IGGPC'14). Le troisième joueur est l'algorithme GRAVE de [3], qui implémente la technique *Generalized Rapid Action Value Estimation technique*, une généralisation de la méthode RAVE [6] adaptée pour GGP. Enfin, nous comparons aussi notre joueur à CFR [20], le *CounterFactual Regret technique*, qui recherche des équilibres de Nash et les suit afin de jouer. CFR est considéré ici comme l'état de l'art pour les jeux GDL-II avec informations incomplètes car l'algorithme HyperPlayer-II [19] n'était pas présentement utilisable au cours des expérimentations.

Pour l'ensemble des matchs, nous utilisons les paramètres de la compétition Tiltyard Open 2015 (la dernière compétition internationale de GGP) : 180 secondes avant le début du jeu et 15 à chaque tour pour jouer.

Afin de comparer notre algorithme MAC-UCB-SYM avec sa dernière version MAC-UCB, nous réalisons une analyse de sensibilité pour résoudre le dilemme entre exploitation (la partie résolution) et exploration (la partie évaluation) mais aussi avec le temps dédié à la détection de symétrie pour MAC-UCB-SYM.

Dans nos expérimentations, 75 % du temps de délibération est dédié à l'exploitation et 25 % à l'exploration pour MAC-UCB. Pour MAC-UCB-SYM, 50 % du temps délibération est dédié à l'exploitation, 25 % à l'exploration et 25 % à la détection de symétrie. Afin de justifier ces ratios, la Figure 2 rapporte l'analyse de sensibilité des 2 algorithmes en moyenne sur l'ensemble des jeux sélectionnés. Le premier graphique indique le score obtenu en moyenne par MAC-UCB contre chaque joueur pour un ratio évoluant entre 0 % et 100 % pour la partie de résolution. L'optimale est atteint entre 72 et 78 %. Le second graphique présente la même valeur pour MAC-UCB-SYM. La partie concernant l'exploitation (UCB) est fixée à 25 % et le temps dédié à la détection de symétrie évolue entre 0 % et 75 %. Le meilleur score obtenu en moyenne est atteint entre 23 % et 27 %.

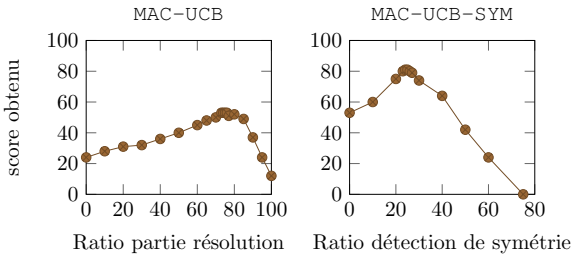


FIGURE 2 – Analyse de sensibilité pour MAC-UCB et MAC-UCB-SYM permettant de paramétrer l'exploitation, l'exploration et la détection de symétrie.

Pour l'ensemble des jeux GDL-I, nous réalisons 100 matchs entre MAC-UCB-SYM et chacun des autres joueurs. Le nombre de matchs pour les jeux GDL-II a été fixé à 400. Par un soucis d'équitabilité, les roles des joueurs sont échangés à chaque match.

Nos résultats sont résumés dans le Tableau 2. Les lignes sont regroupées en 4 parties, respectivement répartissant les jeux GDL-I, les jeux GDL-II sans règles *sees*, les jeux GDL-II en solo avec règles *sees* et les jeux GDL-II à informations partagées (avec les mêmes règles *sees* pour tous les joueurs). La colonne moy :Nogoods indique le nombre moyen de *nogoods* trouvé au cours de l'analyse de la microstructure complément de chaque μSCSP. La colonne #Sym représente le nombre moyen d'automorphismes détectés par MAC-UCB-SYM. La colonne Ame est le ratio entre les tailles des arbres de recherche entre MAC-UCB-SYM et MAC-UCB. Finalement, les dernières colonnes indiquent le pourcentage moyen de victoires de MAC-UCB-SYM contre l'adversaire sélectioné. Par

4. http://sanchoggp.github.io/sancho-ggp/

GDL-I								
Jeu	**moy :Nogoods**	**#Sym**	**Ame**	**MAC-UCB**	**UCT**	**CFR**	**Sancho**	**GRAVE**
Amazons torus 10x10	193,759.44	256	84.50 %	81	96	95	81	76
Battlebrushes	1309,262.32	850	77.01 %	46	76	56	47	53
Breakthrough suicide	24,591.15	124	84.54 %	92	93	95	78	58
Chess	304,993.64	328	83.97 %	77	94	90	87	86
Connect Four 20x20	685,938.7	616	91.02 %	86	98	96	96	62
Connect Four Simultaneous	132,192.04	104	65.16 %	74	89	64	81	74
Copolymer with pie	192,525.23	34	70.24 %	74	87	88	76	80
Dots and boxes suicide	115,705.92	224	59.68 %	63	96	95	86	80
English Draughts	310,185.31	216	82.60 %	84	98	98	58	66
Free For All 2P	51,889.93	58	19.35 %	52	84	82	71	59
Hex	655,047.62	840	71.18 %	84	100	100	79	72
Knight Through	718,259.40	112	71.35 %	81	97	95	87	80
Majorities	473,112.80	670	75.85 %	83	97	94	84	80
Pentago	467,806.70	69	15.97 %	52	84	73	52	70
Quarto	548,426.11	52	12.28 %	53	86	71	56	64
Shmup	137547.01	32	33.33 %	56	86	76	51	53
Skirmish zero-sum	212,782.04	378	83.02 %	84	98	99	67	79
TicTac Chess 2P	661,546.67	143	76.64 %	95	98	94	85	74
TTCC4 2P	701,739.72	84	86.61 %	82	97	98	68	60
Reversi Suicide	334,927.41	256	61.36 %	71	100	100	57	62
GDL-II sans règles *sees*								
Jeu	**moy :Nogoods**	**#Sym**	**Ame**	**MAC-UCB**	**UCT**	**CFR**	**Sancho**	**GRAVE**
Backgammon	272,034.67	318	88.40 %	91.2	98.7	94.1	100	68
Can't Stop	630,456.94	847	89.06 %	88.5	93.5	88.5	100	92
Kaseklau	86,342.57	384	58.68 %	72.1	69.4	78.4	86.4	67
Pickomino	142,420.03	248	84.50 %	74.6	73.7	71.1	91.4	86.4
Yahtzee	256,497.69	340	91.48 %	86.7	86.4	89.4	90.4	62.4
GDL-II en solo avec des règles *sees*								
Jeu	**moy :Nogoods**	**#Sym**	**Ame**	**MAC-UCB-SYM**	**MAC-UCB**	**UCT**	**CFR**	**GRAVE**
GuessDice	0	0	0.00 %	15.6	15	15.7	16	16.5
MasterMind	4	4	0.00 %	63.9	67.8	53.8	68.1	60.1
Monty Hall	0	0	0.00 %	65.1	65.2	62.5	63.1	64.3
Vaccum Cleaner Random	164,875.58	324	83.79 %	74.8	61.5	34	46	58.8
Wumpus	369,117.69	128	78.95 %	71.2	32.1	40.1	44.1	51.2
GDL-II avec la même règle *sees* pour tous								
Jeu	**moy :Nogoods**	**#Sym**	**Ame**	**MAC-UCB**	**UCT**	**CFR**	**GRAVE**	
Pacman	81,704.31	244	20.63 %	62.3	61.2	64.2	60.4	
Schnappt Hubi	137,845.47	314	94.10 %	79.3	84.4	76.2	74.7	
Sheep & Wolf	54,729.44	212	61.00 %	84.1	90.2	74.3	73.2	
TicTacToe Latent 3P 10x10	919,414.21	678	79.23 %	84.5	95.1	91.7	84.6	
War (card game)	32,294.47	51	56,80 %	63.2	71.3	68.4	62.4	

TABLE 2 – Les résultats de MAC-UCB-SYM.

exemple, la case de la colonne UCT pour les échecs (la 4ème ligne) montre, que MAC-UCB-SYM gagne en moyenne dans 94% des matchs contre UCT. Sachant que le 3ème groupe ne comportent que des jeux à un joueur, les colonnes des joueurs (incluant MAC-UCB-SYM) rapportent le nombre moyen de fois où le jeu est résolu divisé par le nombre total de matchs.

Les résultats en GDL-I. Concernant les jeux déterministes, MAC-UCB-SYM surpasse clairement sa version sans symétrie. Ceci peut s'expliquer par la significative amélioration obtenue dans le nombre de noeuds explorés par MAC-UCB-SYM : une amélioration de plus de 80 % est observé dans les plus gros jeux, indiquant que la détection de symétrie est inestimable pour de tels jeux. Les quelques exceptions peuvent s'expliquer d'elles-même. Le *Free For All 2P*, le *Pentago* et le *Quarto* sont des jeux dont l'arbre de recherche n'est pas très important et où les symétries ne sont pas fréquentes ce qui explique la faible amélioration d'exploration de l'arbre de recherche sur MAC-UCB. Concernant les jeux à 4 joueurs : Le *Battlebrushes* dont l'objectif est de peindre le plus de cases possibles en 20 tours avant les 3 autres adversaires, ici la meilleure stratégie est atteinte très rapidement car l'arbre qui est de petite taille tout comme le *Shmup*, un *shoot'em up* possédant un horizon faible de seulement 20 tours. Pour toutes ces exceptions MAC-UCB-SYM obtient un score moyen semblable à son prédécesseur car l'arbre de recherche est rapidement complètement exploré par les algorithmes GGP. UCT et CFR sont vaincus par MAC-UCB-SYM, avec parfois un score obtenu de 100 % (voir le jeu *Hex*). Sancho, le *leader* de IGGPC'14, est en moyenne moins compétitif que MAC-UCB-SYM spécifiquement pour les jeux impliquant un arbre de recherche important. Nous pouvons également noter certains exemples notables comme l'*Amazons torus 10x10*, les échecs et le Puissance Quatre 20x20. Enfin GRAVE est le meilleur rival de MAC-UCB-SYM pour les jeux déterministes, mais notre algorithme obtient un score moyen meilleur pour cette catégorie.

Les résultats en GDL-II. Pour les jeux GDL-II sans règles *sees*, les quatres adversaires sont battus avec un score supérieur à 70 % pour MAC-UCB-SYM contre Sancho, UCT et CFR et supérieur à 60 % contre GRAVE. Il est possible de comprendre ce phénomène car, dans les jeux stochastiques, il est typique de rencontrer de nombreuses symétries sur les états probabilistes générées par la participation d'un joueur "chance" (simulant les dés ou les cartes). Un exemple remarquable est le jeu du *Yahtzee*, où une amélioration de plus de 90 % est observée (contre MAC-UCB), à l'aide de la détection de symétrie.

Pour les jeux avec informations partielles, Sancho ne peut pas participer car il est dédié à GDL-I (modulo une possible simulation du joueur "chance"). Pour les jeux à un joueur avec informations incomplètes, seuls le *Vaccum Cleaner* et le *Wumpus* sont assez grands pour exploiter les symétries. MAC-UCB-SYM est le meilleur joueur pour ces jeux, et réalise une réelle amélioration sur MAC-UCB. Cependant, il est parfois surpassé par CFR sur le *Mastermind* et le *MontyHall*. Notons qu'aucune symétrie n'est détectée dans le jeu *GuessDice* (où il faut en un tour deviner la valeur d'un dé) et au *MontyHall*. De plus les résultats pour le *Mastermind* sont assez peu significatifs suite à la petite taille de l'arbre de recherche.

Enfin, les 5 derniers jeux sont des jeux à informations partielles mais partagées entre les joueurs. Par exemple, le jeu *TicTacToe Latent 3P 10x10* implique 3 joueurs (en incluant le joueur "chance"). Le joueur "chance" place aléatoirement une croix ou un rond sur l'une des cases libres et les 2 autres joueurs ne peuvent le remarquer uniquement en essayant de placer par chance, leur symbole respectif sur ces cases. Le *Schnappt Hubi* (l'attrape fantôme) et le *Sheep & Wolf* sont des jeux coopératifs, où l'ensemble des agents partagent leurs observations dans le but de vaincre le joueur "chance". En dehors du *Pacman*, il est possible d'observer que la détection de symétrie pour les POSGs est aussi payante, caractérisée par une réduction substantielle de l'arbre de recherche. MAC-UCB-SYM gagne avec un score moyen supérieur à 60 % contre chaque adversaire, et particulièrement plus, avec un score moyen de 84 % pour le *TicTacToe Latent Random 10x10*.

6 Conclusion

Dans ce papier, nous avons identifié un fragment important des jeux à informations incomplètes qui peut être représenté en SCSPs, et qui peut être résolu avec des techniques usuelles de la programmation par contraintes. Notre algorithme de décisions séquentielles MAC-UCB-SYM pour les jeux GDL-II exploite la propagation par contraintes, l'évaluation Monte-Carlo et la détection de symétrie. Basé sur de nombreuses expérimentations impliquant de nombreux types de jeux et de joueurs GGP, nous avons montré que les techniques usuelles de programmation par contraintes portent leurs fruits. Notamment, la détection de symétrie offre une amélioration importante pour la résolution des jeux à informations incomplètes.

À la lumière de ces résultats expérimentaux, un axe de recherche est l'étude théorique de la détection de symétries pour des algorithmes basés sur UCB tel que UCT et GRAVE.

Références

[1] Edouard Bonnet and Abdallah Saffidine. On the complexity of general game playing. In *Proceedings of CGW*, pages 90–104, 2014.

[2] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218) :145–149, 2015.

[3] Tristan Cazenave. Generalized rapid action value estimation. In *Proceedings of IJCAI 2015*, pages 754–760, 2015.

[4] Davis Cohen, Peter Jeavons, Christopher Jefferson, Karen E. Petrie, and Barbara M. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints*, 11(2-3) :115–137, 2006.

[5] Florian Geißer, Thomas Keller, and Robert Mattmüller. Past, present, and future : An optimal online algorithm for single-player GDL-II games. In *Proceedings of ECAI*, pages 357–362, 2014.

[6] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *Proceedings of ICML*, pages 273–280, 2007.

[7] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing : Overview of the AAAI competition. *AAAI Magazine*, 26(2) :62–72, 2005.

[8] Michael R. Genesereth and Michael Thielscher. *General Game Playing*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2014.

[9] Matthew L. Ginsberg. GIB : imperfect information in a computationally challenging game. *J. Artif. Intell. Res. (JAIR)*, 14 :303–358, 2001.

[10] Judy Goldsmith and Martin Mundhenk. Competition adds complexity. In *Proceedings of NIPS*, pages 561–568, 2007.

[11] Brahim Hnich, Roberto Rossi, S. Armagan Tarim, and Steven Prestwich. Filtering algorithms for global chance constraints. *Artificial Intelligence*, 189 :69–94, 2012.

[12] Tom Kelsey, Steve Linton, and Colva M. Roney-Dougal. New developments in symmetry breaking in search using computational group theory. In *Proceedings of AISC'04*, pages 199–210, 2004.

[13] Frédéric Koriche, Sylvain Lagrue, Éric Piette, and Sébastien Tabary. General game playing with stochastic CSP. *Constraints*, 21(1) :95–114, 2016.

[14] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General game playing : Game description language specification. Technical report, Stanford University, 2008.

[15] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60(0) :94 – 112, 2014.

[16] Jean François Puget. Automatic detection of variable and value symmetries. In *Proceedings of CP'05*, pages 475–489, 2005.

[17] Stephan Schiffel and Michael Thielscher. Reasoning about general games described in gdl-ii. In *Proceedings of AAAI 2011*, pages 846–851. AAAI Press, 2011.

[18] Stephan Schiffel and Michael Thielscher. Representing and reasoning about the rules of general games with imperfect information. *J. Artif. Intell. Res. (JAIR)*, 49 :171–206, 2014.

[19] Michael John Schofield and Michael Thielscher. Lifting model sampling for general game playing to incomplete-information models. In *Proceedings of AAAI 2015*, pages 3585–3591, 2015.

[20] Mohammad Shafiei, Nathan Sturtevant, and Jonathan Schaeffer. Comparing uct versus cfr in simultaneous games. In *IJCAI Workshop on General Game Playing*, 2009.

[21] Nathan R. Sturtevant. An analysis of uct in multiplayer games. *ICGA Journal*, 31(4) :195–208, 2008.

[22] Michael Thielscher. A general game description language for incomplete information games. In *Proceedings of AAAI'10*, pages 994–999, 2010.

[23] Toby Walsh. Stochastic constraint programming. In *Proceedings of ECAI'02*, pages 111–115, 2002.

Le méta-problème des langages Mal'tsev conservatifs

Clément Carbonnel
LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France
carbonnel@laas.fr

Résumé

Les polymorphismes Mal'tsev définissent une importante famille de langages traitables qui généralisent les équations linéaires. Dans ce papier, nous nous intéressons au méta-problème associé : étant donné un langage de contraintes Γ, Γ admet-il un polymorphisme Mal'tsev ? Bien que nous ne soyons pas en mesure d'établir la complexité de ce problème dans toute sa généralité, nous présentons un algorithme polynomial dans le cas où le polymorphisme est conservatif. Ce papier est un résumé d'un article présenté à AAAI-16 [4].

Contexte L'approche algébrique des problèmes de satisfaction de contraintes consiste à étudier la complexité des langages de contraintes par le biais de leurs polymorphismes, c'est-à-dire des opérations qui préservent les relations. Cette approche a été très fructueuse et a abouti par exemple à la classification complète de la complexité des langages qui contiennent toutes les relations unaires sur leur domaine [3].

Si l'on dispose d'une opération $f : \mathcal{D}^k \to \mathcal{D}$, on peut définir une opération f^* qui agit sur des *tuples* de valeurs en appliquant f composante par composante. On dit alors que f est un polymorphisme d'une relation R si appliquer f^* à n'importe quelle combinaison de k tuples de R produit toujours un tuple de R. Par extension, f est un polymorphisme d'un langage s'il est polymorphisme de chacune de ses relations.

De nombreuses conditions suffisantes pour qu'un langage soit traitable ont été identifiées en termes d'existence d'un ou plusieurs polymorphismes ayant des propriétés particulières ; on peut citer par exemple les polymorphismes de majorité ou les polymorphismes binaires associatifs, commutatifs et idempotents [6].

Langages Mal'tsev Un polymorphisme ternaire est Mal'tsev s'il satisfait $f(y, x, x) = f(x, x, y) = y$ pour tout $x, y \in \mathcal{D}$. Par exemple, l'opération $f(x, y, z) = x - y + z$ est Mal'tsev, et de manière générale toutes les relations équivalentes à une conjonction d'équations linéaires sur un corps fini admettent un polymorphisme Mal'tsev. Il a été démontré que tout langage qui admet un polymorphisme Mal'tsev induit un CSP pouvant être résolu en temps polynomial [2]. Le principe fondamental est qu'une relation Mal'tsev peut être encodée en ne gardant qu'un faible nombre de tuples ; la relation d'origine est alors la clôture de ces tuples par le polymorphisme Mal'tsev. L'algorithme pour résoudre un CSP sur un langage Mal'tsev part d'une instance sans aucune contrainte, puis ajoute les contraintes une par une en gardant à chaque étape une représentation compacte de l'ensemble des solutions (qui est lui-même une relation Mal'tsev).

Méta-problème et uniformité La définition la plus faible d'une classe traitable est un ensemble de langages T tel que pour tout $\Gamma \in T$, $\mathrm{CSP}(\Gamma) \in \mathrm{P}$ (où $\mathrm{CSP}(\Gamma)$ désigne la restriction de CSP aux instances dont les contraintes n'utilisent que des relations de Γ). Suivant cette définition, l'ensemble formé par tous les langages Mal'tsev forme une classe traitable. On peut renforcer cette notion de deux façons.

La première est d'imposer que le méta-problème, qui est le problème de décider si un langage donné appartient à T, doit pouvoir être résolu en temps polynomial. On notera que le méta-problème ne fait aucune hypothèse sur le langage, et en particulier ne suppose pas que le domaine est constant (ce qui est une hypothèse très courante dans la littérature). Dans le pire des cas le méta-problème d'une classe peut être indécidable, mais en pratique il est souvent dans NP. Un méta-problème polynomial est une condition nécessaire pour qu'une classe traitable puisse être utilisable en pratique, par exemple via un prétraitement

des instances dans un solveur CSP.

De façon complémentaire, on peut imposer que la restriction CSP(T) de CSP aux instances dont le langage est dans T forme un problème polynomial. Ce n'est pas toujours vrai, car la définition faible d'une classe traitable implique qu'il existe un algorithme polynomial $\mathcal{A}(\Gamma)$ pour chaque $\Gamma \in T$ fixé, mais ne dit rien sur l'existence d'un algorithme polynomial "unifié" $\mathcal{A}$ qui résout CSP(T). Si un tel algorithme existe, on dit que T est *uniformément traitable*. Par exemple, un algorithme exponentiel dans la taille du domaine pourra être polynomial pour un Γ fixé, mais ne sera pas polynomial si on l'applique à CSP(T).

Problème considéré Pour la classe des langages Mal'tsev, la complexité du méta-problème et la question de l'uniformité sont des problèmes non résolus à l'heure actuelle. L'algorithme décrit précédemment pour résoudre un CSP sur un langage Mal'tsev n'est pas uniforme car il a besoin d'avoir accès à une représentation explicite d'un polymorphisme Mal'tsev qui préserve le langage. Une énumération exhaustive est exclue car il existe $O(d^{d^3})$ opérations Mal'tsev sur un domaine de taille d.

Certains cas particuliers ont cependant été résolus. Par exemple, il est connu que si l'on se restreint aux graphes orientés (langages composés d'une unique relation binaire) alors les langages Mal'tsev sont uniformément traitables et le méta-problème est polynomial [5]. Un autre cas est celui des polymorphismes Mal'tsev *conservatifs*, c'est-à-dire satisfaisant $f(x, y, z) \in \{x, y, z\}$ pour tout $x, y, z \in \mathcal{D}$. Cette famille de polymorphismes est intéressante, car tout langage qui reste traitable en présence d'une contrainte globale de cardinalité a un polymorphisme Mal'tsev conservatif. Elle joue également un rôle crucial dans une preuve récente de la classification des langages conservatifs. Il a été prouvé que les langages Mal'tsev conservatifs *binaires* ont un méta-problème polynomial et sont uniformément traitables [1].

Contribution La contribution principale de ce papier est une extension du résultat pour le cas conservatif aux contraintes d'arité quelconque.

Théorème. *La classe des langages admettant un polymorphisme Mal'tsev conservatif est uniformément traitable et son méta-problème peut être résolu en temps polynomial.*

Les deux parties du thèorème sont prouvées simultanément en donnant un algorithme qui décide si un langage admet un polymorphisme Mal'tsev conservatif f, et produit une représentation explicite de f si c'est le cas. L'algorithme est élémentaire : on encode le problème en CSP, on applique l'arc-cohérence, on utilise une règle simple pour éliminer des variables et on résout l'instance résiduelle par élimination de Gauss. La correction est en revanche assez délicate à prouver et s'appuie sur une analyse détaillée de l'instance CSP.

En plus de cela, nous avons appliqué nos méthodes à une autre classe. Un polymorphisme ternaire f est de *majorité* si pour tout $x, y \in \mathcal{D}$, $f(x, x, y) = f(x, y, x) = f(y, x, x) = x$. Les polymorphismes de majorité définissent une classe uniformément traitable, et le méta-problème est polynomial. Cependant, l'algorithme présenté par Bessière et al. [1] pour le méta-problème est très coûteux : même dans le cas conservatif, il faut $O(rlt^4d^6)$ opérations pour un langage de l relations d'arité au plus r, composées d'au plus t tuples et sur un domaine de taille d. En adaptant nos méthodes, nous avons obtenu un algorithme de complexité $O(rlt^4)$ pour le cas conservatif.

Malheureusement, nos preuves sont très dépendantes de l'hypothèse de conservativité. Il faudra donc développer de nouveaux outils théoriques pour résoudre le cas général.

Question 1. *Quelle est la complexité du méta-problème pour les langages Mal'tsev ? Cette classe est-elle uniformément traitable ?*

Références

[1] Christian Bessiere, Clément Carbonnel, Emmanuel Hebrard, George Katsirelos, and Toby Walsh. Detecting and exploiting subproblem tractability. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 468–474. AAAI Press, 2013.

[2] Andrei A. Bulatov. Mal'tsev constraints are tractable. Technical report, Computing Laboratory, University of Oxford, Oxford, UK, 2002.

[3] Andrei A. Bulatov. Tractable conservative constraint satisfaction problems. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science*, pages 321–330, Ottawa, Canada, 2003.

[4] Clément Carbonnel. The meta-problem for conservative Mal'tsev constraints. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[5] Catarina Carvalho, László Egri, Marcel Jackson, and Todd Niven. On Maltsev digraphs. In *Computer Science–Theory and Applications*, pages 181–194. Springer, 2011.

[6] Peter Jeavons, David A. Cohen, and Marc Gyssens. Closure properties of constraints. *J. ACM*, 44(4) :527–548, 1997.

Extraction d'un sous-ensemble maximal d'informations qui soit cohérent avec des contextes mutuellement contradictoires

Éric Grégoire **Yacine IZZA** * **Jean-Marie Lagniez**

CRIL, Université d'Artois & CNRS, F-62307 Lens, France
{gregoire,izza,lagniez}@cril.fr

Résumé

L'extraction efficace d'un sous-ensemble maximal d'informations qui soit cohérent avec des contextes ou des sources d'informations multiples est un problème clé dans de nombreux domaines de l'Intelligence Artificielle ; en particulier, lorsque ces contextes ou sources peuvent être mutuellement contradictoires. Dans cette étude, cette question est adressée d'un point de vue calculatoire pratique en logique propositionnelle. Une nouvelle approche est proposée et nos expérimentations montrent qu'elle surpasse les techniques de l'état de l'art. Une version en langue anglaise de cet article a été publié à la conférence AAAI 16 [8].

Abstract

The efficient extraction of one maximal information subset that does not conflict with multiple contexts or additional information sources is a key basic issue in many Artificial Intelligence. domains, especially when these contexts or sources can be mutually conflicting. In this paper, this question is addressed from a computational point of view in clausal Boolean logic. A new approach is introduced that experimentally outperforms the currently most efficient technique. An English version of this paper has been published in AAAI 16 [8].

1 Introduction

L'extraction efficace d'un sous-ensemble maximal d'informations cohérent avec des contextes multiples ou des sources d'informations supplémentaires est un problème clé dans de nombreux domaines de l'Intelligence Artificielle, en particulier, lorsque ces contextes ou sources peuvent être mutuellement contradictoires. Comme illustré dans [5], ce problème est central en planification, dans la prise de décision, le diagnostic, l'argumentation et le raisonnement non monotone, ainsi que dans bien d'autres champs de l'Intelligence Artificielle.

Considérons par exemple un agent qui doit prendre des décisions extrêmement rapidement à partir de ses propres informations. Il souhaite être prudent et ne prendre que des décisions qui soient compatibles avec un éventail d'hypothèses prospectives, possiblement contradictoires, qu'il pourrait envisager au sujet de ce qu'il ne sait pas. En raison des contraintes de temps, il cherche à extraire rapidement un sous-ensemble maximal d'informations qui ne contredit aucune de ces hypothèses et considère ce sous-ensemble comme formant une base acceptable pour sa prise de décision. De cette manière, chaque hypothèse restera compatible avec ses propres décisions et la prise en compte supplémentaire de n'importe quelle autre de ses propres prémisses entrainera une contradiction avec au moins une de ces hypothèses. Cette extraction pourrait être aussi à la base d'une méthode d'énumération de tous les sous-ensembles maximaux cohérents avec toutes les hypothèses lorsque ces sous-ensembles demeurent en petit nombre et qu'un temps de calcul suffisant est disponible. Elle peut également être adaptée de façon à respecter un ordre de préférence entre informations.

De manière générale, l'extraction d'un sous-ensemble maximal d'informations cohérent avec des sources d'informations supplémentaires, sources pouvant être mutuellement contradictoires, peut jouer un rôle dans les systèmes multi-agents et la négociation, car cela peut consister à trouver un sous-ensemble maximal qui n'entre en conflit avec aucun des objectifs fixés par les agents ou les négociateurs.

Dans cette étude, cette question est étudiée d'un point de vue calculatoire pratique en logique propositionnelle, lorsque la maximalité se réfère à l'inclusion ensembliste.

*Papier doctorant : Yacine IZZA est auteur principal.

Une nouvelle approche est introduite et les expérimentations menées montrent qu'elle surpasse les méthodes existantes.

L'article est organisé comme suit. Dans la section 2, nous rappelons des notions élémentaires de logique propositionnelle ainsi que des concepts utiles pour la lecture de l'article. Puis, nous précisons les définitions formelles du problème étudié dans la section 3. Dans les sections 4 et 5, nous présentons l'état de l'art pour ensuite introduire progressivement notre nouvelle approche en sections 6 et 7. La section 8 est dédiée à l'étude comparative et expérimentale de ces méthodes. Enfin, nous concluons dans la section 9.

2 Préliminaires techniques

Soit $\mathcal{L}$ le langage de formules de la logique propositionnelle standard, construit sur un ensemble énumérable de variables booléennes, notées $a, b, c, \ldots$ Les symboles $\neg, \vee, \wedge$ et $\Rightarrow$ représentent respectivement les connecteurs de la négation, de la disjonction, de la conjonction et de l'implication matérielle. Un littéral est une variable booléenne ou sa négation. Une clause est une disjonction de littéraux, notée $\alpha, \beta, \gamma, \ldots$ Un ensemble de clauses est noté $\Delta, \Sigma, \ldots$ Un ensemble d'ensembles de clauses est noté $\mathcal{C}, \mathcal{D}, \ldots$ La cardinalité d'un ensemble Δ s'écrit *card*(Δ). Une interprétation est une fonction qui assigne à chaque variable une valeur de vérité *vrai* ou *faux*. Un ensemble de clauses Δ est *satisfiable* ssi il existe au moins une interprétation (appelée alors modèle) qui satisfait toutes les clauses de Δ. SAT est un problème NP-complet qui consiste à vérifier si un ensemble fini de clauses est satisfiable [7].

Dans la suite, nous considérerons que Δ est un ensemble fini de clauses, que Φ et Ψ sont des sous-ensembles de Δ et que $\mathcal{C}$ est un ensemble fini de contextes, notés Γ_i, tels que chaque Γ_i est un ensemble satisfiable de clauses. L'article s'appuie aussi sur les concepts de MSS et de CoMss.

Définition 1 *Φ est un sous-ensemble maximal$_\subseteq$ satisfiable de Δ (ou* MSS *pour Maximal Satisfiable Subset en anglais) ssi Φ est satisfiable et $\forall \alpha \in \Delta \setminus \Phi$, $\Phi \cup \{\alpha\}$ est unsatisfiable.*

L'ensemble complémentaire d'un MSS dans Δ est un sous-ensemble minimal rectificatif de Δ (en anglais : *Minimal Correction Subset*) appelé également MCS ou CoMSS.

Définition 2 *Ψ est un sous-ensemble minimal rectificatif de Δ, ssi $\Psi = \Delta \setminus \Phi$, où Φ est un* MSS *de Δ.*

En conséquence, Δ peut être toujours partitionné en un couple (MSS, MCS). L'extraction d'une telle partition est intraitable dans le pire cas car ce problème appartient à la classe $FP^{NP}[wit, log]$, c.-à-d., à l'ensemble des problèmes qui peuvent être calculés en un temps polynomial en exécutant un nombre logarithmique d'appels à un oracle NP qui renvoie un certificat pour un résultat positif [14].

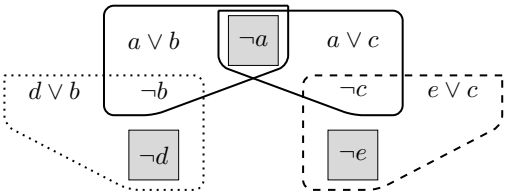


FIGURE 1 – Exemple 1.

Des techniques de calcul d'un MSS/MCS expérimentalement efficaces existent cependant comme par exemple [9, 13, 15]. À noter que dans le pire cas, le nombre de MSSes est exponentiel par rapport au nombre de clauses de Δ.

Un concept qui a attiré l'attention de beaucoup de chercheurs est celui de sous-ensemble minimal insatisfiable (ou MUS pour *Minimal Unsatisfiable Subset*) d'un ensemble contradictoire de clauses Δ. Un MUS de Δ est un sous-ensemble insatisfiable de Δ qui est minimal dans le sens où supprimer toute clause du sous-ensemble le rendrait satisfiable. Une grande quantité de travaux ont été réalisés au sujet de l'extraction de MUS en pratique ; voir par exemple [3, 6, 16, 11, 4, 10, 12].

3 Énoncé du problème

Nous reprenons de [5] les définitions de sous-ensembles maximaux satisfiables et minimaux rectificatifs *sous un ensemble de contextes hypothétiques*.

Définition 3 *Φ est un sous-ensemble maximal satisfiable de Δ sous un ensemble de contextes hypothétiques $\mathcal{C}$, noté* AC-MSS$(\Delta, \mathcal{C})$*, ssi*

1. *Φ est un sous-ensemble satisfiable de Δ ;*
2. *$\forall \Gamma_i \in \mathcal{C}$, $\Phi \cup \Gamma_i$ est satisfiable ;*
3. *$\forall \alpha \in \Delta \setminus \Phi$, $\Phi \cup \{\alpha\}\Gamma_i$ est insatisfiable pour chaque $\Gamma_i \in \mathcal{C}$.*

Exemple 1 *Soit $\Delta = \{a \vee b, a \vee c, d \vee b, e \vee c, \neg b, \neg c\}$ et $\mathcal{C} = \{\{\neg a\}, \{\neg d\}, \{\neg e\}\}$. Soulignons que cet exemple est un cas simple dans le sens où (1) Δ est satisfiable, (2) tous les contextes sont de simples clauses unitaires et (3) les contextes ne sont pas mutuellement contradictoires. Dans le cas général ces trois propriétés n'ont pas besoin d'être satisfaites. Dans cet exemple, la Figure 1 illustre comment les différents contextes de $\mathcal{C}$ entrent en conflit avec les clauses de Δ. Les ensembles de clauses regroupés dans des cercles représentent les MUS formés par $\Delta \cup \mathcal{C}$, avec à chaque fois un contexte de $\mathcal{C}$. Dans cet exemple, $\{a \vee b, a \vee c, d \vee b, e \vee c\}$ constitue un* `AC-MSS`$(\Delta, \mathcal{C})$*, parmi d'autres.*

Définition 4 *Ψ est un sous-ensemble minimal rectificatif de Δ sous un ensemble de contextes hypothétiques $\mathcal{C}$, noté* AC-MSS$(\Delta, \mathcal{C})$*, ssi $\Psi = \Delta \setminus \Phi$, où Φ est un* AC-MSS$(\Delta, \mathcal{C})$.

Exemple 2 *Dans l'exemple 1, $\{a \vee b, a \vee c, d \vee b, e \vee c\}$, $\{a \vee b, d \vee b, \neg c\}$, $\{a \vee c, e \vee c, \neg b\}$ et $\{\neg c, \neg b\}$ sont des* AC-MCS$(\Delta, \mathcal{C})$.

Nous nous focaliserons sur l'extraction soit d'un AC-MSS$(\Delta, \mathcal{C})$, soit d'un AC-MCS$(\Delta, \mathcal{C})$, et ce de manière interchangeable dans cet article. Quand un des ensembles est retourné, l'autre se calcule de manière directe, puisque leur combinaison constitue une partition de Δ.

Dans la suite de l'article, nous utiliserons des approximations de AC-MSS$(\Delta, \mathcal{C})$ et AC-MCS$(\Delta, \mathcal{C})$, définies comme suit.

Définition 5 *Φ est un* sous-ensemble satisfiable *de Δ sous un ensemble de contextes hypothétiques $\mathcal{C}$, noté* AC-SS$(\Delta, \mathcal{C})$*, ssi il existe au moins un ensemble de clauses Φ' t.q. $\Phi' \subseteq \Phi \subseteq \Delta$ et Φ' est un* AC-MSS$(\Delta, \mathcal{C})$

Définition 6 *Ψ est un* sous-ensemble rectificatif *de Δ sous un ensemble de contextes hypothétiques $\mathcal{C}$, noté* AC-CS$(\Delta, \mathcal{C})$*, ssi il existe au moins un ensemble de clauses Ψ' t.q. $\Psi' \subseteq \Psi \subseteq \Delta$ et Ψ' est un* AC-MCS$(\Delta, \mathcal{C})$.

Insistons sur le fait que ni Δ, ni la conjonction de tous les Γ_i de $\mathcal{C}$, ne doivent être forcément satisfiables dans les définitions. Les Γ_i sont considérés un à un : chaque AC-MSS$(\Delta, \mathcal{C})$ doit être satisfiable avec chaque contexte Γ_i, sélectionné individuellement. Dans [5], des exemples illustrent pourquoi les Γ_i ne peuvent pas être remplacés par un contexte globalisé qui soit formé par leur conjonction ou leur disjonction, même dans les cas où aucun Γ_i n'en contredit aucun autre.

4 Approche par transformation

Cette méthode d'extraction d'un AC-MSS$(\Delta, \mathcal{C})$, proposée dans [5], est la plus performante qui existe dans la littérature. Elle fait appel à une procédure de prétraitement (`Greedy-AC-SS`) qui permet d'extraire un sous-ensemble d'un AC-MSS$(\Delta, \mathcal{C})$, c.-à-d., un AC-SS$(\Delta, \mathcal{C})$ de départ qu'il faudra maximiser par la suite. Typiquement, `Greedy-AC-SS` consiste à prendre un ensemble de clauses Δ' initialisé à Δ, lequel va être réduit de manière itérative jusqu'à ce que Δ' devienne satisfiable avec tous les Γ_i. En effet, à chaque itération, un Γ_i est considéré, les clauses satisfiable avec Γ_i sont gardées et les autres sont ainsi supprimées.

Exemple 3 *Soit $\Delta = \{\neg a \vee b, \neg b, b \vee d\}$ et $\mathcal{C} = \{\Gamma_1 = \{a\}, \Gamma_2 = \{\neg d\}\}$. `Greedy-AC-SS` se déroule comme suit. Supposons que Γ_1 soit considéré en premier et $\neg a \vee b$ soit sélectionné et retiré de Δ à cette étape. Ensuite, admettons que $\neg b$ soit supprimé afin de rendre Δ' satisfiable avec Γ_2. L'ensemble final $\Delta' = \{b \vee d\}$ n'est pas un* AC-MSS$(\Delta, \mathcal{C})$*. En effet, seule la suppression de $\neg b$ de Δ permettra d'obtenir un plus grand ensemble $\{\neg a \vee b, b \vee d\}$, lequel est un* AC-MSS$(\Delta, \mathcal{C})$.

Dans [5], une approche de *force brute* d'extraction d'un AC-MSS$(\Delta, \mathcal{C})$ est analysée. Cette approche consiste à calculer pour chaque Γ_i tous les sous-ensembles maximaux de Δ cohérents avec Γ_i, avant d'extraire la solution finale qui minimise les clauses écartées. Le problème majeur de cette méthode est le nombre potentiellement exponentiel de MSS qu'il peut y avoir à calculer, rendant pareille approche intraitable le plus souvent.

Pour pallier cette difficulté, du moins dans une certaine mesure, les auteurs ont proposé dans [5] un schéma d'encodage qui permet de réécrire le problème en un seul calcul de MSS avec quelques appels de plus à un solveur SAT. Précisément, quand m est le nombre de Γ_i dans $\mathcal{C}$, n' le nombre maximal de clauses dans un Γ_i, n le nombre de clauses dans Δ, la méthode nécessite dans le pire cas $O(m)$ appels à un solveur SAT sur une instance de taille $O(n + n')$, plus un nombre logarithmique d'appels à un solveur SAT sur une instance de taille initiale $O(m(n + n'))$ qui est divisée par deux à chaque appel. Sans surprise, l'`Approche par Transformation` s'avère plus efficace que la méthode *force-brute* dans beaucoup d'instances. Cependant, de très grandes valeurs pour m demeurent problématiques sur le plan pratique car la taille de l'instance transformée est proportionnelle à m.

5 Algorithme incrémental basique

Algorithme 1 : `Incremental`$_1$`-AC-MSS`

Entrée : Δ : un ensemble de clauses;
$\mathcal{C} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_m\}$: un ensemble d'ensembles de clauses satisfiables;
Sortie : Φ s.t. Φ est un `AC-MSS`$(\Delta, \mathcal{C})$;

```
1 Φ ← Greedy-AC-SS(Δ, C);  Ψ ← (Δ \ Φ);
2 pour chaque α ∈ Ψ faire
3     all-Γ_i-satisfied ← true;   i ← 1;
4     tant que i ≤ n et all-Γ_i-satisfied faire
5         all-Γ_i-satisfied ← Φ ∪ {α} ∪ Γ_i est
          satisfiable;
6         i ← i + 1;
7     si all-Γ_i-satisfied alors  Φ ← Φ ∪ {α};
8 retourner Φ;
```

En effet, il existe une autre technique que celle par transformation et *brute-force* pour parititionner Δ en un AC-MSS$(\Delta, \mathcal{C})$ et un AC-MCS$(\Delta, \mathcal{C})$.

`Incremental`$_1$`-AC-MSS` est décrit dans l'Algorithme 1. Initialement, il appelle la procédure `Greedy-AC-SS` qui renvoie Φ, l'AC-MSS$(\Delta, \mathcal{C})$ en construction. Ensuite, Φ est augmenté par les clauses α

de $\Delta \setminus \Phi$ tel que, à chaque itération, $\Phi \cup \{\alpha\}$ est satisfiable avec tous les Γ_i. La validité de cet algorithme se démontre facilement à partir des trois points de la Définition 3 : la satisfiabilité de Φ est garantie à chaque étape ; par construction, l'ensemble final Φ ne contredit aucun Γ_i et pour chaque clause α de $\Delta \setminus \Phi$ il existe au moins un Γ_i pour lequel $\Phi \cup \{\alpha\} \cup \Gamma_i$ est unsatisfiable.

Exemple 4 *Déroulons* `Incremental`$_1$`-AC-MSS` *sur l'Exemple 1. Supposons que* $\{a \vee b, a \vee c, e \vee c\}$ *soit renvoyé par* `Greedy-AC-SS`*.* Φ *est initialisé à cet ensemble et ainsi* $\Psi = \{d \vee b, \neg c, \neg b\}$ *(ligne 1). Chaque clause de* Ψ *est alors considérée successivement. Quand* $\alpha = d \vee b$*,* $\Psi \cup \{\alpha\}$ *est satisfiable avec chaque contexte et ainsi* α *est ajouté à* Φ*. Quand* $\alpha = \neg c$ *(et quand* $\alpha = \neg b$*), comme* $\Phi \cup \{\alpha\}$ *est insatisfiable,* $\neg a$ *n'est pas ajouté à la solution en construction.*

Indépendamment du coût de calcul de la procédure `Greedy-AC-SS`, `Incremental`$_1$`-AC-MSS` effectue mn appels à un solveur SAT dans le pire cas, où m et n représentent respectivement *card*$(\mathcal{C})$ et *card*(Δ).

Algorithme 2 : `Greedy-AC-SS`

Entrée : Δ : un ensemble de clauses ;
$\mathcal{C}$: un ensemble d'ensembles de clauses satisfiables;
Sortie : $\Phi \subseteq \Delta$ t.q. Φ est un `AC-SS`$(\Delta, \mathcal{C})$;

1 $\Psi \leftarrow \Delta$; $\Phi \leftarrow \emptyset$; $cpt \leftarrow 0$;
2 **répéter**
3 $\Upsilon \leftarrow \Psi$;
4 **pour chaque** $\Gamma_i \in \mathcal{C}$ **faire**
5 Soit I un modèle de $\Gamma_i \cup \Phi$;
6 $\Upsilon \leftarrow \Upsilon \cap \{\alpha \in \Psi$ s.t. $I(\alpha) = true\}$;
7 $\Phi \leftarrow \Phi \cup \Upsilon$; $\Psi \leftarrow \Psi \setminus \Upsilon$; $cpt \leftarrow cpt + 1$;
8 **jusqu'à** $\Upsilon = \emptyset$ ***or*** $cpt >$ #max-iteration;
9 **retourner** Φ;

Nous avons utilisé une version plus efficace de `Greedy-AC-SS` (voir l'algorithme 2) dans nos expérimentations. Celle-ci consiste à partitionner progressivement Δ en un couple (Φ, Ψ), où Φ est le `AC-SS`$(\Delta, \mathcal{C})$ courant et Ψ est l'`AC-CS`$(\Delta, \mathcal{C})$ courant. Pour chaque Γ_i, un modèle I de $\Gamma_i \cup \phi$ est calculé. Les clauses de Ψ satisfaites par I sont sauvegardées dans Υ ; Φ et Ψ sont mis à jour, les clauses de Υ sont déplacées de Ψ vers Φ. La recherche des interprétations initiales est réalisée en appellant un solveur SAT sur Δ. Comme dans [9], la technique dite "*progress saving* interpretation" est utilisée pour la recherche d'un modèle de $\Gamma_i \cup \Phi$ car elle permet de satisfaire un plus grand nombre de clauses de Δ.

Plus tard, nous parlerons des performances de l'algorithme `Incremental`$_1$`-AC-MSS` muni de ce prétraitement. À présent, nous allons présenter notre nouvelle méthode incrémentale, plus élaborée et beaucoup plus performante que l'actuelle.

6 Propriétés utiles

Pour construire des algorithmes incrémentaux plus efficaces, nous nous sommes focalisés sur des propriétés qui nous permettent d'identifier les clauses à ajouter au AC-MSS$(\Delta, \mathcal{C})$ ou au AC-MCS$(\Delta, \mathcal{C})$, pendant leurs constructions. À partir de maintenant, nous travaillerons sur l'ensemble AC-MCS$(\Delta, \mathcal{C})$ qui forme le sujet de notre algorithme original.

Intuitivement, la première propriété se présente comme suit.

Lorsqu'on sait que l'ajout d'une clause α *à l'AC-CS en construction rendra l'AC-SS cohérent avec tous les contextes et non pas seulement avec une partie d'entre eux, alors il est approprié d'insérer* α *dans l'AC-CS courant. En effet, dans ces cas là, il existe forcément un AC-MCS au moins qui est inclus dans l'AC-CS courant contenant* α*. De plus, tous ces AC-MCS contiennent nécessairement* α*.* Formellement :

Propriété 1 *Admettons que* $\Theta \subset \Delta$ *et que* $\alpha \in (\Delta \setminus \Theta)$*.*

Si $\exists\, \mathcal{C}'$ *t.q.* $\mathcal{C}' \subseteq \mathcal{C}$ *et* $\exists\, \Gamma_j \in \mathcal{C}'$ *t.q.* $\forall\, \Gamma_i \in \mathcal{C} \setminus \mathcal{C}'$

1. $(\Delta \setminus \Theta) \cup \Gamma_i$ *est satisfiable ;*
2. $(\Delta \setminus \Theta) \cup \Gamma_j$ *est insatisfiable ;*
3. $\forall\, \Gamma_k \in \mathcal{C}'$, $(\Delta \setminus (\Theta \cup \{\alpha\})) \cup \Gamma_k$ *est satisfiable*

alors il existe au moins un `AC-MCS`$(\Delta, \mathcal{C})$*, nommé* Ψ*, t.q.*
(1) $\Psi \subseteq \Theta \cup \{\alpha\}$*, et*
(2) $\forall \Psi$ *qui sont* `AC-MCS`$(\Delta, \mathcal{C})$ *t.q.* $\Psi \subseteq \Theta \cup \{\alpha\}$: $\alpha \in \Psi$*.*

Intuitivement, pour comprendre pourquoi cette propriété tient, remarquons simplement que (1) tient en partie, comme l'exprime la condition 3, au fait que la suppression de α de Δ permettra de satisfaire les contextes précédemment insatisfiables et ceux qui ne sont pas encore pris en compte, que (2) est facilement compréhensible par raisonnement par contradiction. Supposons qu'en même temps $\exists\, \Psi$ tel que Ψ soit un `AC-MCS`$(\Delta, \mathcal{C})$, $\Psi \subseteq \Theta \cup \{\alpha\}$ et $\alpha \notin \Psi$. Par conséquent, $\Psi \subseteq \Theta$ et grâce à la condition 2, nous avons $(\Delta \setminus \Psi) \cup \Gamma_j$ qui est insatisfiable, ce qui contredit l'hypothèse que Ψ est un `AC-MCS`$(\Delta, \mathcal{C})$.

Exemple 5 *Considérons* Δ *et* $\mathcal{C}$ *de l' Exemple 1. Admettons que* $\Theta = \{a \vee b, d \vee b, \neg b\}$*,* $\mathcal{C}' = \{\{\neg a\}, \{\neg e\}\}$*,* $\alpha = \neg c$ *et* $\Gamma_j = \{\neg a\}$*. Toutes les conditions sont satisfaites pour appliquer la Propriété 1 . Ainsi,* α *appartient à tous les* `AC-MCS` *pouvant être obtenus à partir de* Δ *et* $\mathcal{C}$ *et qui sont inclus dans* $\{a \vee b, d \vee b, \neg b, \neg c\}$ *(c.-à-d.,* $\{\{a \vee b, b \vee d, \neg c\}, \{\neg b, \neg c\}\}$*).*

Nous avons cherché à généraliser cette propriété dans le but de développer un algorithme qui extrait un AC-MCS$(\Delta, \mathcal{C})$. La notation suivante à la Partial-MaxSAT nous sera utile pour établir la propriété généralisée.

Définition 7 *Soit Ω_1 et Ω_2 deux ensembles de clauses, où Ω_1 est satisfiable. La procédure* `Partial-MCS`(Ω_1, Ω_2) *fournit un ensemble minimal (par rapport à l'inclusion ensembliste) de clauses Σ de Ω_2 t.q. $\Omega_1 \cup (\Omega_2 \setminus \Sigma)$ est satisfiable. Les clauses de Ω_1 sont dites dures et les clauses de Ω_2 sont dites souples. Pour des soucis de notations, nous utiliserons le même nom de cette procédure pour exprimer le résultat retourné.*

Notons que `Partial-MCS`(Ω_1, Ω_2) est une variante d'une procédure d'extraction d'un MCS et peut être implantée en tant que telle : elle peut naturellement bénéficier des récents progrès réalisés dans le calcul de MCS (voir par exemple : [9, 15]).

La généralisation s'établit comme suit. On peut remplacer α dans la Propriété 1 par n'importe quel `Partial-MCS`$((\Delta \setminus \Upsilon) \cup \Gamma_j, \Upsilon \setminus \Theta)$. *Intuitivement, on peut étendre l'AC-CS en construction, en lui ajoutant un ensemble minimal de clauses qui permet de rendre l'AC-SS correspondant satisfiable avec un context supplémentaire, à condition que l'AC-CS étendu rende l'AC-SS correspondant satisfiable avec tous les contextes.*

Propriété 2 *Supposons que Υ soit un* `AC-CS`$(\Delta, \mathcal{C})$, $(\Theta \cup \Sigma) \subseteq \Upsilon$ *et $\Theta \cap \Sigma = \emptyset$.*

Si $\exists\, \mathcal{C}'$ t.q. $\mathcal{C}' \subseteq \mathcal{C}$ et $\exists\, \Gamma_j \in \mathcal{C}'$ t.q. $\forall\, \Gamma_i \in \mathcal{C} \setminus \mathcal{C}'$

1. $(\Delta \setminus \Theta) \cup \Gamma_i$ *est satisfiable ;*
2. $(\Delta \setminus \Theta) \cup \Gamma_j$ *est insatisfiable ;*
3. Σ *est un* `Partial-MCS`$((\Delta \setminus \Upsilon) \cup \Gamma_j, \Upsilon \setminus \Theta)$ *;*
4. $\forall\, \Gamma_k \in \mathcal{C}'$, $(\Delta \setminus (\Theta \cup \Sigma)) \cup \Gamma_k$ *est satisfiable*

donc il existe au moins un Ψ qui est un `AC-MCS`$(\Delta, \mathcal{C})$*t.q.*

(1) $\Psi \subseteq \Theta \cup \Sigma$, et
(2) $\forall \Psi$ s.t. Ψ est un `AC-MCS`$(\Delta, \mathcal{C})$ *et $\Psi \subseteq \Theta \cup \Sigma$:$\Sigma \subseteq \Psi$.*

Exemple 6 *Considérons à nouveau Δ et $\mathcal{C}$ de l'Exemple 1. Soit $\Theta = \{d \vee b\}$, $\mathcal{C}' = \{\{\neg a\}, \{\neg e\}\}$, $\Sigma = \{\neg c, \neg b\}$ et $\Gamma_j = \{\neg a\}$. Toutes les conditions de la Propriété 2 sont satisfaites. Ainsi, Σ apparait dans chaque* `AC-MCS`$(\Delta, \mathcal{C})$ *inclus dans $\{d \vee b, \neg b, \neg c\}$.*

Dans la prochaine section, Nous verrons comment cette propriété sera exploiter pour améliorer l'extraction d'un `AC-MCS`$(\Delta, \mathcal{C})$.

7 Algorithme incrémental plus élaboré

Un algorithme original pour l'extraction d'un `AC-MCS`$(\Delta, \mathcal{C})$ est illustré dans l'Algorithm 3, que nous allons décrire progressivement et de façon intuitive.

Par abus de notation, nous écrirons Σ résout Γ_j quand la Propriété 2 s'applique (en référence à Σ et Γ_j de la Propriété 2) : dans ce cas Σ est un sous-ensemble de clauses de Δ qui sera inclus dans le AC-MCS final, en cours de construction.

Ψ est le AC-MCS en construction : au départ, il est initialisé à l'ensemble vide (line 1). La structure $\sigma(\Gamma_i)$ associe à chaque Γ_i un sous-ensemble de Δ. Initialement, i est affecté à 1 et ces sous-ensembles à l'ensemble vide, à l'exception de $\sigma(\Gamma_1)$, qui est initialisé à $\Delta \setminus$ `Greedy-AC-SS`$(\Delta, \mathcal{C})$. Pour simplifier la compréhension, on considère pour le moment qu'il n'y a pas de pré-traitement et donc à la ligne 4, on a $\sigma(\Gamma_i) \leftarrow \Delta$.

Ensuite, on entre dans la boucle principale. Durant le traitement de la boucle, on fait en sorte que $\Psi \cup \bigcup_{\Gamma_i \in \mathcal{C}} \sigma(\Gamma_i)$ soit tout le temps un AC-CS$(\Delta, \mathcal{C})$. Cette contrainte est bien-sûr satisfaite avant d'entrer dans la boucle vu que cet ensemble est égal à Δ. Considérons la première itération de la boucle. $i = 1$ et à la ligne 7, on calcule `Partial-MCS`(Γ_1, Δ) et on stocke le résultat dans Σ. Plusieurs cas se présentent. Si $\Sigma = \sigma(\Gamma_i)$: dans cette première itération, nous avons alors effectivement $\Delta = \Sigma = \sigma(\Gamma_i)$. Dans ce cas de figure, toutes les conditions sont remplies pour appliquer la Propriété 2 et ainsi on peut insérer Σ dans la solution en cours de construction Ψ (ligne 8). On affecte un ensemble vide à $\sigma(\Gamma_i)$ pour signifier que Γ_i a été résolu. i est donc décrémenté et la boucle termine étant donné que la valeur de i est égale à zéro : Ψ est le résultat renvoyé. Dans un autre cas vraiment basique, $i = m = 1$: il y a seulement un seul contexte dans $\mathcal{C}$. De nouveau, nous avons trouvé la solution, c'est Σ ; mais contrairement à la situation précédente, Σ doit impérativement être un sous-ensemble propre de Δ. Dans le dernier cas (*else*), on sait que Σ est l'ensemble de clauses à retirer pour satisfaire Γ_i. Toutefois, certains Γ_j n'ont pas été résolus jusque là et il n'est pas certain que Σ fasse entièrement partie de la solution car la Propriété 2 ne s'applique pas. Par conséquent, on insère $\sigma(\Gamma_i) \setminus \Sigma$ dans $\sigma(\Gamma_{i+1})$ et on sauvegarde Σ dans $\sigma(\Gamma_i)$. L'idée sous-jacente consiste à garder la trace dans $\sigma(\Gamma_i)$ du sous-ensemble de clauses qui aurait été suffisant pour résoudre Γ_i, tandis que les autres clauses *actuellement* restantes vont être considérées par rapport à Γ_{i+1}. Dans un certain sens, les éléments de σ sont poussés d'un cran vers l'avant. De cette manière, les clauses restantes seront considérées à la prochaine itération. Tandis qu'au même moment, on enregistre dans $\sigma(\Gamma_i)$ l'ensemble des clauses qui pourraient rendre Γ_i satisfiable avec l'AC-MSS en construction avec $\bigcup_{k=1}^{i-1} \sigma(\Gamma_k)$.

Á l'inverse, à ligne 7 quand la Propriété 2 s'applique, i est décrémenté. Cela conduit donc à une sorte de mouvement en arrière : Γ_i a été résolu, on peut à présent revenir en arrière et traiter Γ_{i-1} à la prochaine itération. Tout cela est réalisé de telle manière que $\sigma(\Gamma_j) = \emptyset$ pour tout $j > i$ au début et à la fin de chaque itération.

À présent, nous sommes dans une meilleure position pour comprendre le contenu de la boucle *while* et la première instruction figurant à l'intérieur (ligne 6). $\Delta \setminus (\Psi \cup \bigcup_{j=1}^{i} \sigma(\Gamma_j))$ fournit l'AC-SS en cours d'élaboration. Par construction, il est satisfiable avec Γ_i. Maintenant, `Partial-MCS`$((\Delta \setminus (\Psi \cup \bigcup_{j=1}^{i} \sigma(\Gamma_j))) \cup \Gamma_i, \sigma(\Gamma_i))$ fournit le plus petit sous-ensemble de toutes les clauses restantes accumulées dans $\sigma(\Gamma_i)$, qui doit être supprimé de $\sigma(\Gamma_i)$ pour que ce dernier soit satisfiable avec le premier argument de `Partial-MCS`. L'objectif est d'arriver à la situation où toutes les clauses $\sigma(\Gamma_i)$ doivent être retirées, c.-à-d., $\Sigma = \sigma(\Gamma_i)$ (ligne 7), et dans ce cas, toutes les conditions sont réunies pour appliquer la Propriété 2. En effet, on a trouvé $\Sigma =$ `Partial-MCS`$((\Delta \setminus \Upsilon) \cup \Gamma_i, \Upsilon \setminus \Theta)$ tel que l'extension de l'AC-SS courant avec Σ le rende satisfiable avec tous les Γ_i, étant donné qu'il ne reste plus de clauses à tester dans $\sigma(\Gamma_i)$. Quand $i = m$, la même opération peut être appliquée : clairement, en rendant l'AC-SS courant satisfiable avec Γ_m, on est certain que tous les Γ_i sont résolus. Dans les deux cas, on peut décrémenter i : à la prochaine itération, on tentera de résoudre Γ_{i-1} pour lequel des clauses ont été enregistrées dans $\sigma(\Gamma_{i-1})$. En effet, quand la Propriété 2 ne s'applique pas, on mémorise dans $\sigma(\Gamma_i)$ le sous-ensemble de clauses calculé pour résoudre Γ_i (mais qui ne permet pas de résoudre les autres Γ_j à la fois) et mettre les clauses restantes dans σ_{i+1} pour être considérées à la prochaine itération où on traitera Γ_{i+1}. À noter que quand $\sigma(\Gamma_i) = \emptyset$, l'appel à `Partial-MCS` renvoie un ensemble vide et ainsi i est décrémenté (ligne 10).

Exemple 7 *Déroulons* `Incremental`$_2$`-AC-MCS` *sur l'Exemple 1, où* $\Gamma_1 = \{\neg a\}$, $\Gamma_2 = \{\neg e\}$ *et* $\Gamma_3 = \{\neg d\}$. *Soit* $\{a \vee b, a \vee c, e \vee c\}$, *l'AC-SS renvoyé par* `Greedy-AC-SS`. *Avant d'entrer dans la boucle, nous avons* $i = 1$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c, d \vee b\}$ *et* $\sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.

- *Itération* #1 *:* $\Sigma = \{\neg b, \neg c\}$. *Puisque* $\Sigma \neq \sigma(\Gamma_1)$ *et* $i \neq 3$, *on rentre dans la partie* `else`. *Ainsi,* $i = 2$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$, $\sigma(\Gamma_2) = \{d \vee b\}$ *et* $\sigma(\Gamma_3) = \emptyset$.
- *Itération* #2 *:* $\Sigma = \emptyset$. *Idem à la première itération, la partie* `else` *est considérée. Ainsi,* $i = 3$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$, $\sigma(\Gamma_2) = \emptyset$ *et* $\sigma(\Gamma_3) = \{d \vee b\}$.
- *Itération* #3 *:* $\Sigma = \emptyset$. *Comme* $i = n$, *la partie* `if` *est exécutée. Ainsi,* $i = 2$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$ *et* $\sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.
- *Itération* #4 *:* $\Sigma = \emptyset$. *Comme* $\sigma(\Gamma_2) = \emptyset$, *la partie* `if` *est considérée. Ainsi,* $i = 1$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$ *et* $\sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.
- *Itération* #5 *:* $\Sigma = \{\neg b, \neg c\}$, $\Sigma = \sigma(\Gamma_1)$. *La partie* `if` *est exécutée. Ainsi,* $i = 0$, $\Psi = \{\neg b, \neg c\}$, $\sigma(\Gamma_1) = \sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.

L'algorithme sort de la boucle et retourne Ψ, *qui est un* `AC-MCS`$(\Delta, \mathcal{C})$.

Algorithme 3 : `Incremental`$_2$`-AC-MCS`

Entrée : Δ : un ensemble de clauses;
$\mathcal{C} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_m\}$: un ensemble d'ensembles de clauses satisfiables;
Sortie : Ψ t.q. Ψ est un `AC-MCS`$(\Delta, \mathcal{C})$;

1 $\Psi \leftarrow \emptyset$;
2 σ une table de hachage qui associe pour chaque $\Gamma_i \in \mathcal{C}$ un ensemble de clauses, initialement tous à vide;
3 $i \leftarrow 1$;
4 $\sigma(\Gamma_i) \leftarrow \Delta \setminus$ `Greedy-AC-SS`$(\Delta, \mathcal{C})$;
5 **tant que** $i > 0$ **faire**
6 $\quad \Sigma \leftarrow$ `Partial-MCS`$((\Delta \setminus (\Psi \cup \bigcup_{j=1}^{i} \sigma(\Gamma_j))) \cup \Gamma_i, \sigma(\Gamma_i))$;
7 $\quad$ **si** $\Sigma = \sigma(\Gamma_i)$ **ou** $i = m$ **alors**
8 $\quad\quad \Psi \leftarrow \Psi \cup \Sigma$;
9 $\quad\quad \sigma(\Gamma_i) \leftarrow \emptyset$;
10 $\quad\quad i \leftarrow i - 1$;
11 $\quad$ **sinon**
12 $\quad\quad \sigma(\Gamma_{i+1}) \leftarrow \sigma(\Gamma_i) \setminus \Sigma$;
13 $\quad\quad \sigma(\Gamma_i) \leftarrow \Sigma$;
14 $\quad\quad i \leftarrow i + 1$;
15 **retourner** Ψ;

Propriété 3 `Incremental`$_2$`-AC-MCS`$(\Delta,\mathcal{C})$ *renvoie toujours un AC-MCS*$(\Delta,\mathcal{C})$. *Dans le pire cas, cette procédure nécessite* $O(nm)$ *appels à* `Partial-MCS`(Ω_1, Ω_2), *où* $n =$ *card*(Δ) *et* $m =$ *card*$(\mathcal{C})$.

Soulignons que la structure de la procédure `Incremental`$_2$`-AC-MCS` permet de profiter des spécificités avantageuses d'un solveur SAT incremental. À cet égard, nous avons implanté `Partial-MCS` en utilisant une technique similaire à celle qui est présentée dans [1] pour `Glucose` [2] (`http://www.labri.fr/perso/lsimon/glucose/`) afin d'obtenir un système qui réutilise autant que possible les recherches effectuées par le solveur SAT pendant les appels précédents (à l'intérieur d'un même appel à `partial-MCS` et d'un appel à l'autre). Soulignons aussi que cette implantation est caractérisée par d'autres optimisations. En particulier, dans la boucle principale, les appels inutiles à `partial-MCS`$(\ldots, \emptyset)$ sont évités. De plus, notre implantation bénéficie des récentes avancées dans le calcul MSS et MCS, telles que celles proposées dans [9] et utilisées dans [5] pour l'`Approche par Transformation`.

8 Expérimentations

Nous avons confronté les approches `Incremental`$_1$`-AC-MSS`, `Incremental`$_2$`-AC-MCS` et l'`Approche par Transformation` de manière pratique. Pour ce faire, nous avons sélectionné un ensemble de 295 benchmarks insatisfiables de la dernière compétition d'extraction de MUS `http://www.cril.univ-artois.fr/SAT11/results/results.php?idev=48`. Ces

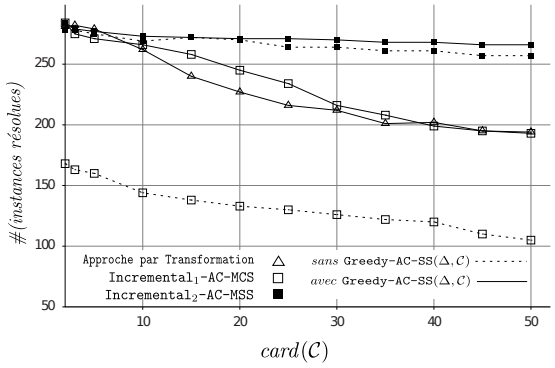


FIGURE 2 – Nombre d'instances résolues.

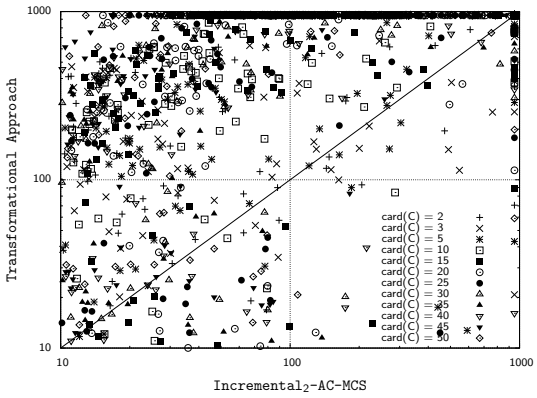


FIGURE 3 – Comparaison des temps de résolution (en secondes) entre `Incremental`$_2$`-AC-MCS` et `Approche de Transformation`.

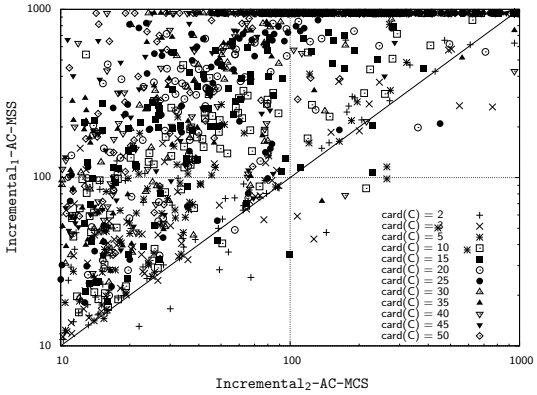


FIGURE 4 – Comparaison des temps de résolution (en secondes) entre `Incremental`$_2$`-AC-MCS` et `Incremental`$_1$`-AC-MSS`.

benchmarks représentent les instances de Δ. Nous avons choisi différentes valeurs pour $card(\mathcal{C})$: $2, 5, 10, 15, 20, 25, 30, 35, 40, 45$ et 50 : ce paramètre représentant les différents nombres de contextes. Puis, en utilisant les variables de Δ, nous avons généré aléatoirement chaque Γ_i, de sorte à avoir un ensemble satisfiable et formé de 50 clauses binaires. Nous avons fait exécuter nos expérimentations sur des processeurs Intel Xeon E5-2643 (3.30GHz) avec 8G octets de mémoire vive et sous Linux CentOS. Nous avons fixé le *timeout* à 900 secondes par instance et par test. Nous avons exécuté l'outil de l'`Approche par Transformation` disponible à l'adresse `http://www.cril.fr/AAAI15-BGL`. Nous avons implanté tous les autres algorithmes en C++ sous `Glucose` (`http://www.labri.fr/perso/lsimon/glucose/`). Notre outil, également que toutes les données et que tous les résultats de ces expérimentations sont disponibles à l'adresse `http://www.cril.fr/AAAI16-GIL`. La constante *max-iteration* dans `Greedy-AC-SS` fut fixée à 10.

Les résultats montrent clairement qu'`Incremental`$_2$`-AC-MCS` est meilleur que les autres approches. Comme l'illustre la Figure 2, pour chaque valeur de cardinalité de $\mathcal{C}$, `Incremental`$_2$`-AC-MCS` est celle qui résout le plus d'instances par rapport aux autres méthodes testées ; la différence croît en fonction de *card(*$\mathcal{C}$*)*. De plus, les résultats ne diffèrent pas lorsque `Incremental`$_2$`-AC-MCS` n'inclut pas `Greedy-AC-SS`. Ce qui montre que les bonnes performances de l'algorithme le sont grâce aux propriétés d'AC-MCS développées dans cette étude. `Incremental`$_1$`-AC-MSS` se révèle moins bon, mais avec l'appel à `Greedy-AC-SS` il arrive à dépasser l'`Approche par Transformation` en termes d'instances résolues. Le nombre d'instances résolues par `Incremental`$_2$`-AC-MCS`, comprises entre 266 et 284 (sur un total de 295 instances testées), dépendent de *card(*$\mathcal{C}$*)*. De même, la plupart des instances sont résolues de manière plus rapide avec `Incremental`$_2$`-AC-MCS`. La Figure 3 compare cette dernière avec l'`approche de Transformation` en terme de temps CPU en secondes écoulées pour résoudre chaque instance, pour chaque valeur de *card(*$\mathcal{C}$*)*. La Figure 4 affiche des résultats similaires quand `Incremental`$_2$`-AC-MCS` et `Incremental`$_1$`-AC-MSS` sont comparés.

Les résultats détaillés de nos expérimentations sont disponibles à l'adresse `http://www.cril.fr/AAAI16-GIL`.

9 Conclusion

La recherche d'un sous-ensemble d'informations satisfiable avec une série de contextes, éventuellement mutuellement contradictoires, est un point central dans beaucoup de domaines de l'Intelligence Artificielle. Dans cet article, nous avons proposé une méthode originale d'extraction d'un tel sous-ensemble en logique propositionnelle. Nos expérimentations montrent que cette méthode obtient les meilleurs résultats, comparée aux approches déjà proposées.

Les résultats obtenus ouvrent la voie à de nombreuses

perspectives. Clairement, la procédure Partial-MCS qui exploite les techniques des solveurs SAT incrémentaux peut aussi être améliorée avec des optimisations spécifiques utilisées par les méthodes d'extraction de MCS, comme celle qui est proposée dans [15]. Également, notre méthode peut constituer la brique élémentaire d'une méthode d'énumération de tous les sous-ensembles d'information maximaux non contradictoire avec une série de contextes possibles (du moins quand cette tâche est réalisable en pratique). La principale difficulté qui réside dans une telle méthode est de savoir comment réutiliser les informations obtenues dans la recherche d'un sous-ensemble, dans l'extraction du prochain sous-ensemble. Enfin, orienter ce travail vers une méthode *incrémentale directe* d'extraction d'un sous-ensemble maximal par rapport à la cardinalité qui soit cohérent avec de multiples contextes demeure un véritable défi.

Références

[1] Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving Glucose for incremental SAT solving with assumptions : Application to MUS extraction. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing - SAT 2013*, volume 7962 of *Lecture Notes in Computer Science*, pages 309–317. Springer, 2013.

[2] Gilles Audemard and Laurent Simon. Lazy clause exchange policy for parallel SAT solvers. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing - SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 197–205. Springer, 2014.

[3] James Bailey and Peter J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Proceedings of the 7th International Symposium on Practical Aspects of Declarative Languages, 7th International Symposium, PADL 2005*, volume 3350 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2005.

[4] Anton Belov, Marijn Heule, and João Marques-Silva. MUS extraction using clausal proofs. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing - SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 48–57. Springer, 2014.

[5] Philippe Besnard, Éric Grégoire, and Jean-Marie Lagniez. On computing maximal subsets of clauses that must be satisfiable with possibly mutually-contradictory assumptive contexts. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, Texas*, pages 3710–3716. AAAI Press, 2015.

[6] Elazar Birnbaum and Eliezer L. Lozinskii. Consistent subsets of inconsistent systems : structure and behaviour. *J. Exp. Theor. Artif. Intell.*, 15(1) :25–46, 2003.

[7] Stephen A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM*, 18(1) :4–18, 1971.

[8] Éric Grégoire, Yacine Izza, and Jean-Marie Lagniez. On the extraction of one maximal information subset that does not conflict with multiple contexts. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 3404–3410, 2016.

[9] Éric Grégoire, Jean-Marie Lagniez, and Bertrand Mazure. An experimentally efficient method for (MSS, CoMSS) partitioning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI'14)*, pages 2666–2673. AAAI Press, 2014.

[10] Éric Grégoire, Bertrand Mazure, and Cédric Piette. Using local search to find msses and muses. *European Journal of Operational Research*, 199(3) :640–646, 2009.

[11] Mark H. Liffiton and Ammar Malik. Enumerating infeasibility : Finding multiple MUSes quickly. In *Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2013*, volume 7874 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2013.

[12] Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning*, 40(1) :1–33, 2008.

[13] João Marques-Silva, Federico Heras, Mikolás Janota, Alessandro Previti, and Anton Belov. On computing minimal correction subsets. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, 2013.

[14] Joao Marques-Silva and Mikolás Janota. On the query complexity of selecting few minimal sets. *Electronic Colloquium on Computational Complexity (ECCC)*, 21 :31, 2014.

[15] Carlos Mencía, Alessandro Previti, and João Marques-Silva. Literal-based MCS extraction. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, 2015*, pages 1973–1979. AAAI Press, 2015.

[16] Alessandro Previti and João Marques-Silva. Partial MUS enumeration. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI Press, 2013.

Clustering conceptuel en PLNE

A. Ouali[12] **S. Loudni**[2] **Y. Lebbah**[1] **P. Boizumault**[2] **A. Zimmermann**[2] **L. Loukil**[1]

[1] Université d'Oran 1, Laboratoire LITIO, 31000 Oran, Algeria.
[2] Université de Caen Normandie, CNRS, UMR 6072 GREYC, 14032 Caen, France.
abdelkader.ouali@unicaen.fr

Résumé

Le clustering conceptuel combine deux tâches classiques de l'apprentissage automatique : le clustering non supervisé d'objets similaires et leur description via des concepts symboliques. Dans cet article [1], nous proposons une approche en 2 étapes en séparant la recherche des descriptions (extraction des motifs fermés) de la formation des clusters (modélisée et résolue à l'aide de la programmation linéaire en nombres entiers). Les expérimentations menées sur différents jeux de données de l'UCI montrent l'efficacité de notre approche pour traiter des problèmes de grande taille et fournir des clusterings de haute qualité.

Abstract

Conceptual clustering combines two long-standing machine learning tasks : the unsupervised grouping of similar instances and their description by symbolic concepts. In this paper, we decouple the problems of finding descriptions and forming clusters by first mining formal concepts (i.e. closed itemsets), and searching for the best k clusters that can be described with those itemsets. Most existing approaches performing the two steps separately are of a heuristic nature and produce results of varying quality. Instead, we address the problem of finding an optimal constrained conceptual clustering by using integer linear programming techniques. Most *other* generic approaches for this problem tend to have problems scaling. Our approach takes advantageous of both techniques, the general framework of integer linear programming, and high-speed specialized approaches of data mining. Experiments performed on UCI datasets show that our approach efficiently finds clusterings of consistently high quality.

1. Une version en langue anglaise de cet article a été acceptée à l'IJCAI'16 : *Efficiently Finding Conceptual Clustering Models with Integer Linear Programming.*

1 Introduction

Le clustering est l'une des tâches fondamentales en fouille de données. L'identification de groupes de données homogènes est une tâche importante, que ce soit pour la classification non-supervisée, la réduction de données ou la détection d'anomalies. Le clustering conceptuel [13, 5, 17, 16] fournit un aspect additionnel : la description des clusters par des concepts symboliques. Les approches classiques [13, 5] combinent la formation des clusters et des descriptions pour arriver à un compromis. Des approches plus récentes [17, 16] ont choisi de séparer la recherche des descriptions, soit avant ou après la formation des clusters.

Étant donné la taille de l'espace de recherche à parcourir, toutes ces approches sont de nature heuristique dont les résultats sont fortement influencés par les conditions d'initialisation, et nécessitent généralement de nombreuses exécutions, entraînant une augmentation des temps de calcul. Pour finaliser le choix du clustering, l'utilisateur devra, de plus, identifier le résultat pertinent. Cette étape est illustrée dans la section 6 consacrée aux expérimentations.

Notre approche consiste à faire appel, successivement, à deux techniques exactes : (1) extraction des motifs fermés (en utilisant l'algorithme LCM [20]) ; (2) sélection des meilleurs clusters (selon un critère d'optimisation) à l'aide de la Programmation Linéaire en Nombres Entiers (PLNE). Nous obtenons ainsi une solution optimale à la problématique du clustering conceptuel.

En PLNE, ce problème a été abordé dans le cadre du *problème de partitionnement et du problème de recouvrement* [15], appliqués à des tâches de clustering dans [9], et adaptés au clustering sous contraintes dans [14]. Les techniques exactes, comme celles décrites dans [8], ont des problèmes de passage à l'échelle, et limitent le nombre de clusters à une valeur constante. En effet,

notre approche basée sur un modèle PLNE formulé sur les motifs fermés, a plus de flexibilité dans l'expression des contraintes, et permet de faire appel aux solveurs performants de la PLNE.

Par rapport à la littérature du problème du clustering conceptuel, notre approche a deux apports majeurs, directement liés à l'usage de la PLNE : exactitude et flexibilité. Notre formulation PLNE modélise toute la problématique en un problème d'optimisation linéaire sous contraintes, dont la résolution par l'algorithmique par séparation/évaluation produit une solution exacte, donnant ainsi des garanties sur la qualité des clusterings que les approches heuristiques.

L'article est organisé comme suit : la section 2 définit le contexte. La section 3 présente la modélisation PLNE, tandis que la section 4 traite des aspects résolution. Un état de l'art synthétique est proposé à la section 5. Les expérimentations menées sont décrites à la section 6. Enfin, nous concluons en ouvrant plusieurs directions de recherche.

2 Contexte et définitions

Dans cette section, nous présentons les définitions et les concepts utilisés au long de cet article.

2.1 Concepts formels

Soit $\mathcal{I}$ un ensemble de n littéraux distincts appelés items, un motif (ou *itemset*) est un sous-ensemble non nul de $\mathcal{I}$. Le langage de motifs correspond à $\mathcal{L}_\mathcal{I} = 2^\mathcal{I} \backslash \emptyset$. Une *base de transactions* est un multi-ensemble de m motifs du $\mathcal{L}_\mathcal{I}$. Chaque motif, généralement appelé une *transaction* ou un objet, est une entrée de la base de transactions. Pour l'instant, la table 1a illustre une base de transactions $\mathcal{T}$ avec m=11 de transactions $t_1, \ldots, t_{11}$ décrites par n=8 d'items A, B, C, D, E, F, G, H.

Soit R une relation binaire entre l'ensemble $\mathcal{T}$ de transactions et l'ensemble $\mathcal{I}$ d'items telle que $(t, i) \in R$ si la transaction t contient l'item i : $i \in t$. On note $\mathcal{R} = (\mathcal{T}, \mathcal{I}, R)$ le tuple formé par ces ensembles de transactions, les items et la relation binaire. R est appelée un **contexte formel** [6]. Deux opérateurs sont définies :

— Soit $I \subseteq \mathcal{I}, ext(I) = \{t \in \mathcal{T} | \ \forall i \in I, (t, i) \in R\}$

— Soit $T \subseteq \mathcal{T}, int(T) = \{i \in \mathcal{I} | \ \forall t \in T, (t, i) \in R\}$

$ext(I)$ est l'ensemble de transactions qui contiennent tous les items dans I. $int(T)$ est l'ensemble des items contenus dans toutes les transactions dans T. Ces deux opérateurs induisent une relation de Galois entre $2^\mathcal{T}$ et $2^\mathcal{I}$, plus formellement, $T \subseteq ext(I) \Leftrightarrow I \subseteq int(T)$.

Une paire telle que $(I = int(T), T = ext(I))$ est appelée un **concept formel**. Cette formulation définit une **propriété de fermeture** sur la base de transactions $\mathcal{T}$, $\texttt{fermé}(I) \Leftrightarrow I = int(ext(I))$. Un motif I pour lequel $\texttt{fermé}(I) =$ vrai est appelé un *motif fermé*.

En utilisant $ext(I)$, nous pouvons définir la *fréquence* d'un concept : $\texttt{freq}(I) = |ext(I)|$, et sa *diversité* : $\texttt{divers}(I) = \sum_{t \in ext(I)} |\{i \in \mathcal{I} \mid (i \notin I) \wedge (i \in t)\}|$. De plus, nous pouvons nous référer à sa *taille* par : $\texttt{taille}(I) = |\{i \mid i \in I\}|$.

2.2 Clustering conceptuel

Le clustering est la tâche qui partitionne un ensemble de transactions en groupes relativement homogènes. Le **clustering conceptuel** consiste à fournir une description distincte de chaque cluster (groupe), c.-à-d. le concept caractérisant l'ensemble des transactions qu'il contient.

Ce problème peut être formulé comme : la recherche d'un ensemble de k clusters, où chacun est décrit par un motif fermé $P_1, P_2, ..., P_k$, et couvrant toutes les transactions sans aucun chevauchement entre les clusters. Par exemple, la table 1c illustre les différents clusterings pour k=3.

Une fonction d'évaluation f est nécessaire pour évaluer la qualité du clustering. Ainsi, le clustering conceptuel cherche un ensemble disjoint de clusters sur $\mathcal{T}$ qui optimisent un certain critère donné sur la qualité du clustering. Par exemple, pour la base de transactions $\mathcal{T}$ et k=3, minimiser $f(P_1, ..., P_k) = \sum_{1 \leq i \leq k} \texttt{divers}(P_i)$ fournit un clustering s_1, avec une valeur optimale 18 (voir la table 1c)

2.3 Autres variantes de clustering

Autres variantes de clustering [3], comme le soft clustering, le co-clustering, et le soft co-clustering, peuvent également être exprimées par la relation entre les transactions et les motifs fermés.

a. Le co-clustering consiste à trouver k clusters couvrant à la fois l'ensemble de transactions et l'ensemble des items, sans aucun chevauchement sur les transactions ou sur les items. Par exemple, s_1 fournit un co-clustering sur la base de transactions $\mathcal{T}$ (voir la table 1c)

b. Le soft clustering consiste à relâcher :

— soit la relation de couverture : les transactions ne sont pas toutes couvertes, mais au moins δ_t transactions doivent être couvertes, où le seuil $\delta_t \leq |\mathcal{T}| (= m)$

— ou la relation de non-chevauchement : de petits chevauchements sont autorisés, mais la transaction t ne peut apparaître dans plus de δ_o clusters, où le seuil $\delta_o \geq 1$

c. Le soft co-clustering consite à relâcher la relation de couverture ou la relation de non-chevauchement

Trans.	Items							
t_1	A	B		D				
t_2	A				E	F		
t_3	A				E		G	
t_4	A				E		G	
t_5		B			E		G	
t_6		B			E		G	
t_7			C		E		G	
t_8			C		E		G	
t_9			C		E			H
t_{10}			C		E			H
t_{11}			C			F	G	H

(a) La base de transactions $\mathcal{T}$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
t_1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
t_2	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0
t_3	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	1	0	1
t_4	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	1	0	1
t_5	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	1
t_6	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	1
t_7	0	0	0	0	0	0	0	1	1	1	0	0	1	0	1	1	0	1
t_8	0	0	0	0	0	0	0	1	1	1	0	0	1	0	1	1	0	1
t_9	0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	0	0	0
t_{10}	0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	0	0	0
t_{11}	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	0	1	1

(b) $(a_{t,c})$ la matrice associée à la base de transactions $\mathcal{T}$.

Sol.	P_1	P_2	P_3
s_1	{A, B, D}	{C, F, G, H}	{E}
s_2	{B}	{C}	{A, E}
s_3	{A}	{C}	{B, E, G}

(c) Trois clusterings conceptuels pour k=3.

TABLE 1 – Exemple typique.

sur l'ensemble de transactions (comme pour le soft-clustering), mais aussi sur l'ensemble des items :

- les items ne sont pas tous couverts, mais au moins γ_i items doivent être couverts, où le seuil $\gamma_i \leq |\mathcal{I}|(= n)$
- de petits chevauchements sont autorisés, mais un item i ne peut apparaître dans plus de γ_o co-clusters, où le seuil $\gamma_o \geq 1$

Example 1 *Pour* δ_t=6, δ_o=1, γ_i=7, γ_o=2, $s_r = [\{A, E\}, \{B, E, G\}, \{C, F, G, H\}]$ *est un soft co-clustering puisque* D *est le seul item manquant, les items* E *et* G *apparaissent deux fois et* $\{t_1, t_7, t_8, t_9, t_{10}\}$ *sont les transactions non couvertes.*

2.4 Programmation linéaire en nombres entiers

La programmation linéaire en nombres entiers (PLNE) [15] est l'une des méthodes les plus utilisées pour le traitement des problèmes d'optimisation, en raison de sa rigueur, sa flexibilité et sa grande capacité de modélisation. Un PLNE est un programme linéaire avec une restriction d'intégralité des variables. En règle générale, un modèle PLNE comporte : (i) un ensemble de variables de décision, (ii) un ensemble de contraintes linéaires, où chaque contrainte est décrite par une expression linéaire portant sur les variables de décision et égale, inférieure ou supérieure à une valeur scalaire, et (iii) une fonction objectif qui permet d'évaluer la qualité de la solution. Résoudre un problème PLNE consiste à trouver la meilleure solution relativement à la fonction objectif dans l'ensemble des solutions faisables. Formellement, un problème PLNE prend la forme :

$$\begin{array}{rl} \texttt{Maximize or Minimize} & c^T x \\ \texttt{Subject to} & Ax\ (\leq, =, or\ \geq)\ b \\ & x_i \in \mathbb{Z}, i = 1..n \end{array} \quad (1)$$

où x représente le vecteur des variables de décision, n est le nombre total de variables entières, c_j ($1 \leq j \leq n$) est le coefficient de la variable x_j dans la fonction objectif. A est une matrice de coefficients de dimension $m \times n$, et b est un vecteur de dimension $m \times 1$ contenant les valeurs du côté droit des contraintes.

3 Modélisations en PLNE

Dans la présente section, nous allons décrire les modèles PLNE correspondants aux différents problèmes de clustering introduits dans la section précédente.

3.1 Clustering conceptuel

Soit $\mathcal{T}$ un ensemble de m transactions définies sur un ensemble $\mathcal{I}$ de n items. Soit $\mathcal{C}$ l'ensemble des p motifs clos (relativement à la mesure de fréquence). Soit $a_{t,c}$ la matrice binaire de dimension $m \times p$ telle que $(a_{t,c} = 1)$ ssi $c \subseteq t$, i.e., la transaction t appartient au cluster représenté par le motif clos c.

Par conséquent, le problème de clustering conceptuel peut être modélisé en un PLNE (voir Fig. 1a) de p variables booléennes $x_c, (c \in \mathcal{C})$, où $(x_c = 1)$ ssi le cluster représenté par le motif clos c appartient au clustering. Les contraintes (1) stipulent que toute transaction t doit être couverte par un et un seul cluster c. La contrainte (2) détermine le nombre $k = k_0$ de clusters. La fonction objectif est définie en associant à chaque cluster $c \in \mathcal{C}$ une valeur v_c qui reflète l'intérêt (à maximiser) ou le coût (à minimiser) du cluster c, par exemple, minimiser la diversité de chaque concept ou maximiser leur taille.

Example 2 *Considérons l'ensemble* $\mathcal{T}$ *des transactions de la Table 1a. La matrice* $(a_{t,c})$ *associée à* $\mathcal{T}$ *est illustrée par la Table 1b. Si on considère* v_c=`size`(c) *la*

valeur attachée au cluster c, alors la solution qui maximise la fonction objectif est x_2=x_{12}=x_{15}=1, *toutes les autres variables* x_c *étant égales à* 0 *(d'où,* k=3*). Cette solution correspond au clustering* s_1 *(voir Table 1c).*

Les mesures de qualité introduites sont associées individuellement aux motifs donnés. La contrainte (2') permet de contrôler le nombre k de clusters en spécifiant une borne inférieure k_{min} et/ou une borne supérieure k_{max}. La relaxation de k lève les contraintes sur le clustering permettant ainsi de fournir des résultats à partir d'un large espace de solutions, ce qui octroie à notre approche une plus grande flexibilité.

3.2 Co-clustering

Le co-clustering consiste à trouver k clusters couvrant à la fois l'ensemble des transactions et l'ensemble des items, sans chevauchements sur les transactions et sur les items. Soit $w_{i,c}$ la matrice binaire $n \times p$ définie comme suit : $(w_{i,c} = 1)$ ssi l'item i appartient au motif clos c. Fig. 1b donne le modèle PLNE du co-clustering. Les contraintes (3) indiquent que tout item i doit être couvert par exactement un cluster c.

Example 3 *La solution optimale* x_2=x_{12}=x_{15}=1 *et toutes les autres variables* x_c *égales à* 0*, correspondant au clustering* s_1 *(voir Table 1c) est un co-clustering puisque les clusters associés couvrent à la fois* $\mathcal{I}$ *et* $\mathcal{T}$ *sans chevauchements.*

3.3 Soft clustering et soft co-clustering

Pour le soft clustering conceptuel (voir Section 2.3b), on introduit m variables booléennes $y_t, (t \in \mathcal{T})$ où $(y_t = 1)$ ssi la transaction t appartient à au moins un cluster. Fig. 1c décrit le modèle PLNE. Premièrement, comme $\sum_{t\in\mathcal{T}} y_t$ donne le nombre de transactions couvertes, la contrainte (5) stipule qu'au moins δ_t transactions doivent être couvertes. Deuxièmement, comme une transaction t appartient à $\sum_{c\in\mathcal{C}} a_{t,c}x_c$ clusters, la contrainte (4) déclare que toute transaction t doit appartenir à au plus δ_o clusters. Noter que si y_t=0, alors la transaction t n'est pas couverte et la contrainte (4) est satisfaite.

Nous procédons de la même manière pour le soft co-clustering (voir Section 2.3c) en introduisant n variables booléennes $z_i, (i \in \mathcal{I})$ où $(z_i = 1)$ ssi l'item i appartient à au moins un cluster. Tout d'abord, comme $\sum_{i\in\mathcal{I}} z_i$ indique le nombre d'items couverts, la contrainte (7) stipule qu'au moins γ_i items doivent être couverts. Ensuite, comme un item i appartient à $\sum_{c\in\mathcal{C}} w_{i,c}x_c$ clusters, la contrainte (6) exprime que tout item i doit appartenir à au plus γ_o clusters. Noter que si z_i=0, alors l'item i n'est pas couvert et la contrainte (6) est satisfaite. L'exemple 1 donne un exemple de soft co-clustering.

Des contraintes additionnelles peuvent être ajoutées au modèle pour éviter des clusterings non pertinents. On peut, par exemple, ajouter des contraintes sur la fréquence des motifs clos (par exemple $\texttt{freq}(c) > 1$) qui conduit à un clustering plus équilibré pour l'ensemble des transactions $\mathcal{T}$, et peut éviter que les données aberrantes soient contenues dans des clusters uniques pour chacune. Dans ce cas, la solution optimale de notre exemple illustratif est x_4=x_7=x_8=1, et toutes les autres variables x_c sont égales à 0. Elle correspond au clustering conceptuel s_3 (voir Table 1c).

4 Résolution

Nous avons défini dans la section précédente un modèle PLNE pour exprimer différents types de clusterings. La puissance de cette plate-forme réside dans sa flexibilité en termes de contraintes pouvant être imposées. Dans la section 3, nous avons défini des contraintes qui imposent *des restrictions globales* sur les clusters telles que : les clusters couvrent toutes les transactions (ou un nombre minimal d'entre elles), les clusters doivent être disjoints (ou ont des chevauchements restreints), etc. Mais en pratique, il peut aussi y avoir des *contraintes locales* qui devraient être définies sur chaque motif clos telles que :
- des contraintes de fréquence *minimale* et *maximale*,
- $\texttt{MustLink}(t_i, t_j)$ qui impose que les transactions t_i et t_j soient dans le même cluster,
- $\texttt{CannotLink}(t_i, t_j)$ qui impose que la transaction t_i ne soit pas dans le même cluster que la transaction t_j.

Plus il y a de contraintes locales sélectives, moins il y aura de motifs clos pour l'étape de résolution. Finalement, le processus de clustering se résume aux deux principales étapes suivantes :
Etape de prétraitement : calcule les motifs clos utilisant des extracteurs efficaces tels que `LCM` et filtre tous ceux qui ne satisfont pas les contraintes locales. Les contraintes telles que la fréquence minimale et de taille des clusters sont directement prises en charge par l'extracteur. Pour les contraintes $\texttt{CannotLink}(t_i, t_j)$, tout cluster c tel que t_i et t_j appartiennent à $ext(c)$ doit être filtré. Les contraintes `MustLink` sont gérées de façon similaire.
Etape de résolution : résout l'un des modèles PLNE de clustering détaillés dans la Section 3 à l'aide de solveurs PLNE efficaces.

5 Travaux relatifs

Le clustering conceptual a été introduit dans [12], et étendu dans [13, 5]. De nombreuses approches ont été conçues pour attaquer ce problème, de nature statistique, syntaxique ou hiérarchique [18], où le

Optimise	$\sum_{c\in\mathcal{C}} v_c \cdot x_c$
Subject to	(1) $\sum_{c\in\mathcal{C}} a_{t,c} \cdot x_c = 1, \quad \forall t \in \mathcal{T}$
	(2) $\sum_{c\in\mathcal{C}} x_c = k_0$
	$x_c \in \{0,1\}, c \in \mathcal{C}$

(a) Modèle PLNE du clustering conceptuel (M1).

Optimize	$\sum_{c\in\mathcal{C}} v_c \cdot x_c$
Subject to	(1) $\sum_{c\in\mathcal{C}} a_{t,c} \cdot x_c = 1, \quad \forall t \in \mathcal{T}$
	(2) $k = \sum_{c\in\mathcal{C}} x_c$
	(2′) $k_{min} \leq k \leq k_{max}$
	(3) $\sum_{c\in\mathcal{C}} w_{i,c} \cdot x_c = 1, \quad \forall i \in \mathcal{I}$
	$k \in \mathbb{N}, \quad x_c \in \{0,1\}, c \in \mathcal{C}$

(b) Modèle PLNE du co-clustering (M2).

Optimize	$\sum_{c\in\mathcal{C}} v_c \cdot x_c$
Subject to	(4) $y_t \leq \sum_{c\in\mathcal{C}} a_{t,c} \cdot x_c \leq \delta_o \cdot y_t, \quad \forall t \in \mathcal{T}$
	(5) $\sum_{t\in\mathcal{T}} y_t \geq \delta_t$
	(6) $z_i \leq \sum_{c\in\mathcal{C}} w_{i,c} \cdot x_c \leq \gamma_o \cdot z_i, \quad \forall i \in \mathcal{I}$
	(7) $\sum_{i\in\mathcal{I}} z_i \geq \gamma_i$
	$k = \sum_{c\in\mathcal{C}} x_c$
	$k_{min} \leq k \leq k_{max}$
	$k \in \mathbb{N},$
	$x_c \in \{0,1\}, \quad c \in \mathcal{C}$
	$y_t \in \{0,1\}, \quad t \in \mathcal{T}$
	$z_i \in \{0,1\}, \quad i \in \mathcal{I}$

(c) Modèle PLNE pour soft co-clustering (M3).

FIGURE 1 – Modèles PLNE du conceptual clustering.

clustering conceptuel hiérarchique est le plus reconnu parmi eux [5, 19].

Approches heuristiques Plusieurs méthodes ont exploré l'idée de découpler la phase de formation de clusters de la phase de recherche des descriptions conceptuelles. Pensa *et al.* [16] ont d'abord commencé par la fouille de motifs (itemsets) clos (ou δ-clos) et de leurs extensions en leur appliquent ensuite le clustering k-Means. Perkowitz et Etzioni [17] font le contraire : leur *fouille de cluster* utilise d'abord une technique de clustering pour former les clusters, puis, exploite ce clustering pour trouver des descriptions à l'aide de l'apprentissage de règles. Les instances sont appariées aux descriptions, conduisant potentiellement à des clusters qui se chevauchent. Enfin, [7] ont introduit le *tiling*. Une *tuile (tile)* est un concept formel et dans ce travail, deux problèmes sont considérés : i) trouver k tuiles qui couvrent le plus de transactions possibles, et ii) trouver le nombre minimal de tuiles qui couvent toutes les transactions. Les deux problèmes sont évidemment des problèmes de clustering conceptuel, les tuiles peuvent se chevaucher, comme pour les clusters.

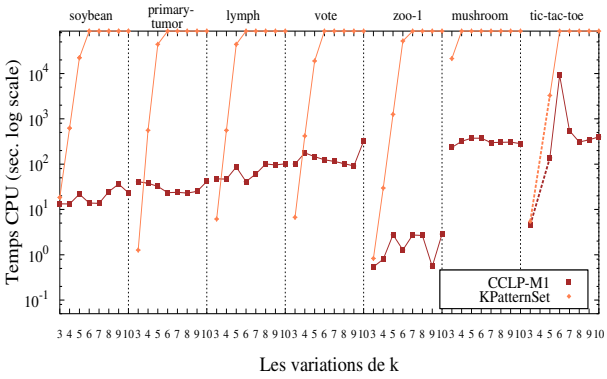
PPC/SAT pour la fouille d'ensemble de motifs. Le paradigme de la programmation par contraintes est au coeur des approches génériques qui traitent le problème des ensembles de k-motifs [10, 8]. Ces méthodes permettent de modéliser les ensembles de motifs satisfaisant des propriétés sur l'ensemble des motifs, de manière déclarative et flexible, comme le clustering conceptuel, le k-tiling, pour ne citer que ceux-là, elles cherchent cependant des ensembles contenant un nombre fixe de motifs et tendent à avoir des problèmes de passage à l'échelle. Dans [11], les auteurs formulent le processus de clustering conceptuel sous forme de requêtes dans un langage donné. Ces requêtes sont traduites en clauses SAT et sont résolues par un solveur SAT. Cependant, la plate-forme proposée ne permet ni d'exprimer des critères d'optimisation ni le passage à l'échelle.

PLNE/PPC pour le clustering basé-distance. Le clustering basé-distance repose sur les dissimilarités (ou similarités) entre des attributs numériques. Dans cet axe, plusieurs approches déclaratives ont été proposées. Dans [4], une approche PPC a été utilisée. Cette approche supporte diverses sortes de contraintes utilisateurs. Dans [2], une approche exacte qui utilise la génération de colonnes pour la résolution d'un PLNE est présentée. Leur approche étend celle proposée par [1]. Dans [14], une approche de clustering sous contraintes utilisant PLNE est proposée. Elle prend en entrée un ensemble de clusters de base possibles et construit un clustering en y sélectionnant un sous-ensemble approprié. Le nombre k de clusters doit être fixé au départ. Notre proposition, en revanche, permet de relâcher k en spécifiant des bornes inférieure et supérieure. Dans leurs expérimentations, l'ensemble des clusters de base est généré à partir des motifs fréquents. Ceci rend l'approche moins applicable car le nombre de clusters de base est énorme et donc la sélection d'un sous-ensemble approprié est difficile. L'adoption de l'ensemble des motifs fermés dans notre approche permet de réduire la redondance comparativement à d'autres modes de sélection des clusters candidats et rend ainsi l'étape de résolution plus efficace en réduisant considérablement le nombre de clusters candidats.

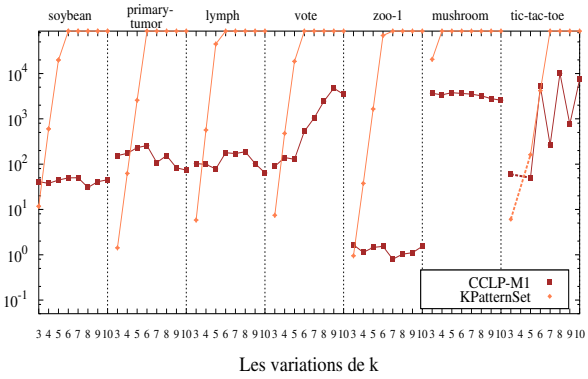
6 Expérimentations

L'évaluation expérimentale est conçue pour répondre aux questions suivantes :

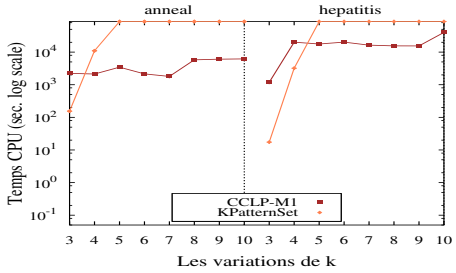
1. Comment (en termes de temps CPU) notre approche se compare par rapport à l'approche



(a) Maximiser la taille de la description.



(b) Minimiser la diversité des motifs.



(c) Maximiser la taille de la description.

Instance	CCLP-M1 avec k relaxé – Maximiser la taille de la description			CCLP-M1 avec k fixé – Maximiser la taille de la description			CCLP-M1-Dist Maximiser la moyenne des similarités du contenu de chaque cluster [14]		
	meilleur	Temps CPU (s.)		meilleur	Temps CPU (s.)		meilleur	Temps CPU (s.)	
	k	(1)	(2)	k	(1)	(2)	k	(1)	(2)
Soybean	10	0.74	15.52	10	0.74	21.31	10	0.74	8.52
Primary-tumor	10	1.98	40.62	10	1.98	35.84	8	1.98	35.4
Lymph	10	3.38	25	10	3.38	93.96	10	3.38	149.69
Vote	10	4.42	191.12	10	4.42	304.18	9	4.42	66.21
Zoo-1	10	0.07	6.22	10	0.07	2.73	9	0.07	0.35
Mushroom	10	17.1	154.75	10	17.1	163.67	10	17.1	169.04
Tic-Tac-Toe	10	0.38	420.75	10	0.38	398.86	10	0.38	209.4
Anneal	10	236.85	1,493.38	10	236.85	5,744.03	10	236.85	5,744.03
Hepatitis	10	113.84	20,825.8	10	113.84	40,014.5	10	113.84	37,745.7

(d) L'impact de la relaxation du nombre de clusters k ((1) étape de prétraitement (2) étape de résolution).

FIGURE 2 – Comparaison des temps CPU.

de Guns et al. (`KPatternSet`) et l'approche de Mueller et al. ?

2. Comment les clusters résultants se comparent par rapport aux différentes fonctions objectif ?
3. Compte tenu de la nature exacte de notre approche, comment les clusters résultants et leurs descriptions se comparent qualitativement avec celles qui résultent des approches heuristiques existantes basées sur une plate-forme similaire ? Les deux approches avec lesquelles nous nous comparons sont celles de Pensa et al. (`CDKMeans`) et Perkowitz et al. (`CM`).

Protocole expérimental. Les expérimentations ont été effectuées sur les mêmes jeux de données utilisés dans [8] et disponibles dans le dépôt de l'UCI. Le tableau 2 présente les caractéristiques de ces jeux de données. Toutes les expérimentations ont été menées sur AMD Opteron 6282SE avec 2.60 GHz de CPU et 512 GB de RAM. Nous avons utilisé `LCM` pour extraire l'ensemble des motifs fermés et `CPLEX` v.12.4

Jeu de données	#transactions	#items	densité(%)	Nombre de motifs clos
Soybean	630	50	32	31,759
Primary-tumor	336	31	48	87,230
Lymph	148	68	40	154,220
Vote	435	48	33	227,031
tic-tac-toe	958	27	33	42,711
Mushroom	8124	119	18	221,524
Zoo-1	101	36	44	4,567
Hepatitis	137	68	50	3,788,341
Anneal	812	93	45	1,805,193

TABLE 2 – Caractérisitque du jeu de données.

pour résoudre les différents modèles PLNE. Au niveau de toutes les méthodes, un Timeout de 24 heures a été fixé.

Les expérimentations ont été réalisées sans contraintes locales sur les motifs fermés. Pour évaluer la qualité d'un clustering, nous avons testé la cohérence d'un clustering, mesurée par la similarité d'intra-cluster *(ICS)*, ainsi que la dissimilarité inter-clusters *(ICD)*, les deux devraient être aussi grandes que possible. La mesure de similarité entre deux transactions t et t' est définie par :

$s : \mathcal{T} \times \mathcal{T} \mapsto [0,1], \;\; s(t,t') = \frac{|t \cap t'|}{|t \cup t'|}$

$$ICS(P_1, ..., P_k) = \frac{1}{2} \sum_{1 \leq i \leq k} (\sum_{t,t' \in P_i} s(t,t'))$$

$$ICD(P_1, ..., P_k) = \sum_{1 \leq i < j \leq k} (\sum_{t \in P_i, t' \in P_j} (1 - s(t,t')))$$

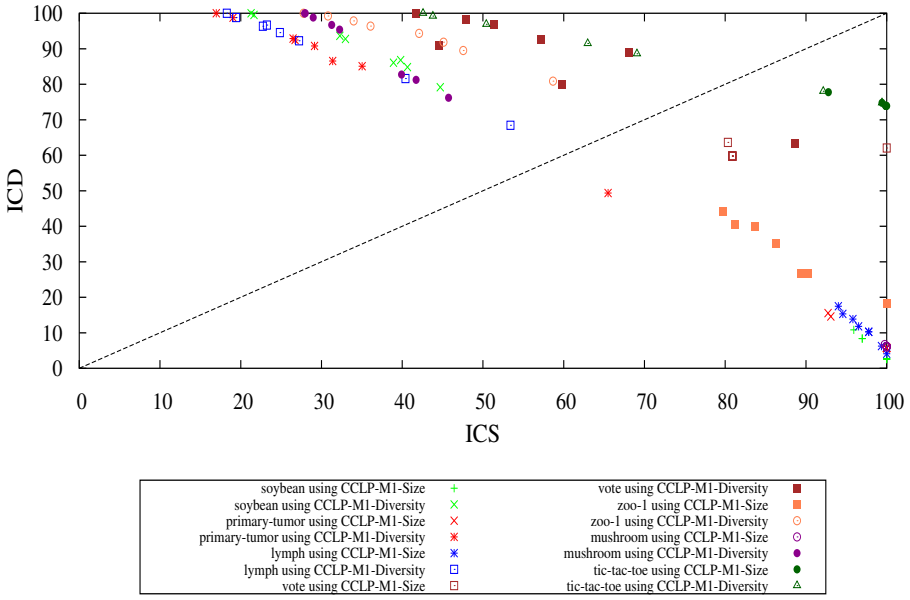
(a) Comparaison de `CCLP` avec `KPatternSet`. Figs. 2a and 2c comparent les performances des deux méthodes en utilisant le modèle `M1` sur différents jeux de données, pour différentes valeurs de k, et par rapport au premier critère d'optimisation (i.e. la taille de la description). Les temps CPU de `CCLP-M1` incluent ceux de l'étape de prétraitement. Pour ($k = 3$), `KPatternSet` domine `CCLP-M1` sauf sur l'instance Soybean, Zoo-1 et Mushroom. Mais pour ($k \geq 4$), `CCLP-M1` est plus performant que `KPatternSet` avec plusieurs ordres de grandeur et sur tous les jeux de données. Enfin, `KPatternSet` ne parvient pas à trouver une solution dans le délai pour ($k > 5$). Sur le jeu de données Audiology, les deux approches ne sont pas en mesure de trouver une solution. Ceci est en partie expliqué par le nombre de motifs fermés (10^6) en comparaison avec les autres jeux de données (allant de 10^3 jusqu'à 10^5). Pour le deuxième critère d'optimisation (see Fig. 2b), les mêmes observations s'appliquent. Notez que sur Tic-Tac-Toe, un clustering conceptuel n'est pas faisable pour (k = 4).

(b) Impact de relaxation de k. Pour évaluer l'intérêt de relâcher k, la Fig. 2d compare, en termes de la meilleure valeur trouvée pour k (Col. 2 vs. Col. 5) et du temps CPU (Col. 4 vs. Col .7), la performance de `CCLP-M1` avec k relâché (`CCLP-M1-relaxed`) contre `CCLP-M1`. Sont rapportées dans Col. 5 les meilleures valeurs trouvées pour k ($3 \leq k \leq 10$) qui maximisent la taille de la description. Les deux configurations obtiennent la même meilleure valeur pour k. En effet, Il y a un biais vers un k près de k_{max} lorsqu'on utilise la taille de la description comme un critère d'optimisation. En comparant les temps CPU, `CCLP-M1-relaxed` est souvent *plus rapide*, notamment sur les deux ensembles de données les plus difficiles Anneal et Hepatitis (speed-up allant jusqu'à 3.84).

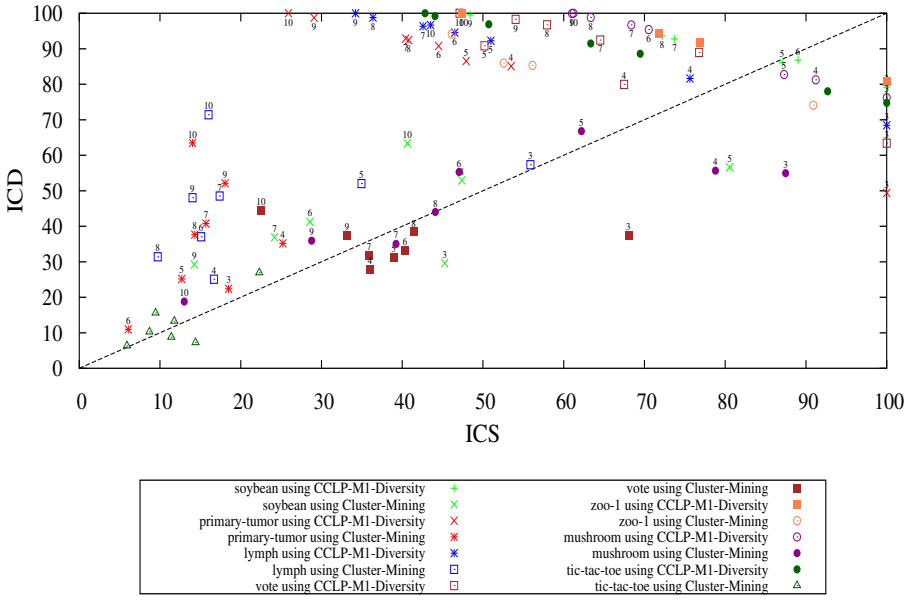
Contrairement à Mueller et al., Notre approche permet de relâcher k. Pour comparer les deux, nous avons implémenté la moyenne des similarités du contenu de chaque cluster (i.e., $s(x,y)$), la fonction objectif utilisée dans leur travail pour un k fixé. Fig. 2d compare, en termes de la meilleure valeur trouvée pour k et le temps CPU, les approches `CCLP-M1-relaxed` et `CCLP-M1-Dist`. Les deux approches retrouvent presque les mêmes meilleures valeurs de k et obtiennent, sur les 7 premiers jeux de données, des temps CPU comparables. Tandis que sur les deux jeux de données difficiles, `CCLP-M1-relaxed` est nettement plus efficace. Fig. 4b montre que les clusterings trouvés en relâchant k sont proches de ceux trouvés avec un k fixé.

Enfin, dans Fig. 2d, on peut voir que l'étape de prétraitement est négligeable par rapport à l'étape de la résolution, à l'exception des deux derniers jeux de données en raison du nombre élevé de motifs fermés.

(c) Analyse qualitative des clusterings. Pour évaluer la qualité des clusterings, nous avons rapporté les mesures de l'*ICS* et de l'*ICD* pour chacun. Les points (voir Fig. 3 et 4) représentent le clustering optimal obtenu pour différentes valeurs de k sur l'ensemble des jeux de données. Une couleur spécifique est attribuée à chaque jeu de données. Une étiquette représentant la valeur de k est associée à certains points avec la même couleur. En outre, nous n'avons pas rapporté les points qui se trouvent sur le front Pareto dans les Figs. 3b et 4a puisqu'ils sont incomparables. Fig. 3a montre les deux critères d'optimisation de `CCLP-M1`. A l'exception des jeux de données Vote et Tic-Tac-Toe, la mesure de la diversité sacrifie l'*ICS* pour atteindre des valeurs de l'*ICD* plus élevées. Ceci est un indicateur sur le choix des clusters plus équilibrés : l'*ICS* est nécessairement limité par le nombre d'instances par cluster, mais l'*ICD* augmente s'il y a plus d'instances dans d'autres clusters. La mesure de taille montre le comportement inverse. Ceci est un indicateur sur le choix des clusters contenant une garde partie des instances et des autres clusters contenant moins d'instances.



(a) `CCLP-M1` : Diversité vs. taille de la description.

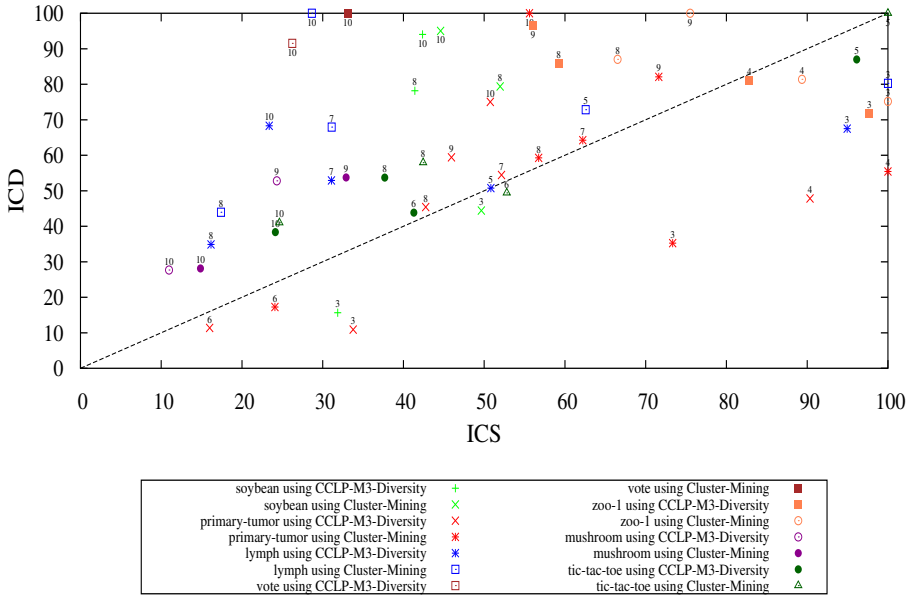


(b) `CCLP-M1` avec la mesure de diversité vs. Cluster Mining (`CM`).

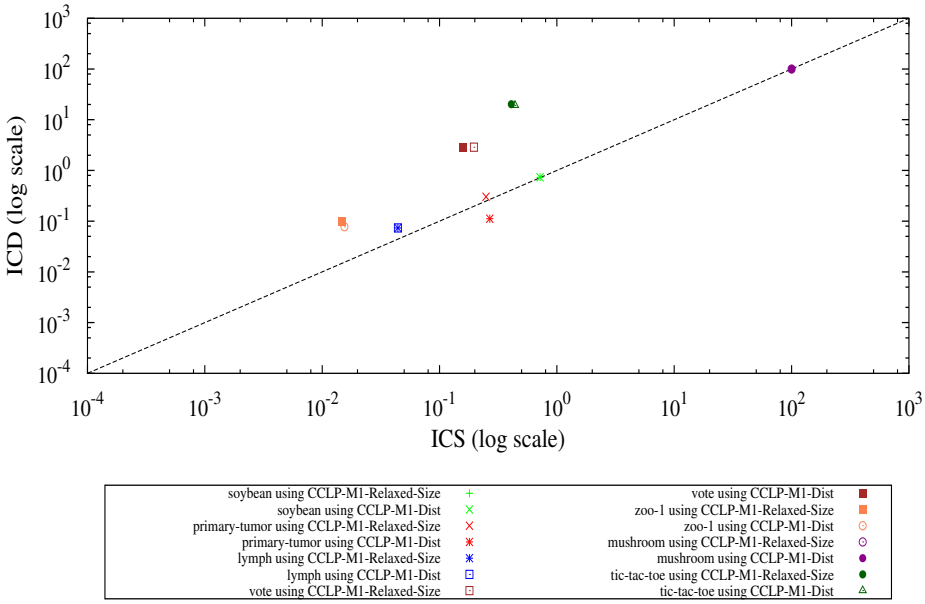
FIGURE 3 – Comparaison de la qualité des clusterings résultants (1).

Pour comparer avec `CDKMeans` et `CM`,[2] nous devons d'abord identifier un bon clustering de comparaison. Pour cela, (*i*) nous avons exécuté chaque technique 100 fois pour lisser les effets d'initialisation aléatoires pour k-Means, (*ii*) puis nous avons regroupé le résultat dans *classes d'équivalence*, dans lesquelles, les clusterings se mettent d'accord sur la composition de *tous* les clusters, (iii) finalement, nous avons choisi la plus grande classe d'équivalence et utilisé son représentant

2. Pour `CM`, nous avons utilisé `SimpleKMeans` et `JRip` dans `Weka`. Nous avons ré-implémenté `CDKMeans` : `http://www.scientific-data-mining.org/software/cdkmeans.zip`.



(a) `CCLP-M3` avec la mesure de diversité vs. Cluster Mining (`CM`).



(b) `CCLP-M1-relaxed` with Size vs. `CCLP-M1-Dist`.

FIGURE 4 – Comparaison de la qualité des clusterings résultants (2).

(à savoir l'un des clusterings en elle, puisque tous ont la même composition). En cas de présence de plusieurs grandes classes d'équivalence, nous avons utilisé un représentant de chaque classe.

Les expérimentations montrent comment il peut être difficile de contrôler les résultats des heuristiques : les clusterings représentatifs de l'approche `CM` ne couvrent jamais toutes les transactions du jeu de données, et ont souvent des clusters qui se chevauchent. Fig. 3b montre que les clusterings calculés par `CM` ont un désavantage qualitatif par rapport aux clusterings trouvés par `CCLP-M1` (les points de `CM` sont toujours do-

minés par celles de CCLP-M1). Afin d'avoir une comparaison équitable, nous avons exécuté CCLP-M3 avec des configurations permettant la même quantité de non-couverture. Si plusieurs grandes classes d'équivalence apparaissent, nous avons choisi le clustering qui maximise *ICS* et *ICD* tout en utilisant sa couverture comme contrainte pour CCLP-M3. La figure 4a montre que notre approche peut avoir des résultats proches des heuristiques.

CDKMeans est encore plus difficile à contrôler par l'utilisateur : lorsque tous les motifs fermés sont utilisés, l'algorithme attribue, généralement, toutes les instances à un ou deux clusters, peu importe la valeur de k. L'utilisation de la contrainte de fréquence améliore un peu cette situation, même si $|\mathcal{C}| \leq k$ pour $k \geq 4$. De plus, les clusterings ne parviennent pas, quelquefois, à se stabiliser finissant ainsi en 100 différents clusterings.

Finalement, la représentation particulière des clusters dans CDKMeans peut entraîner des clusters avec des descriptions sans aucune transaction qui leurs sont assignées.

7 Conclusions

Nous avons proposé une approche efficace pour le clustering conceptuel, qui utilise l'extraction de motifs fermés pour découvrir des candidats pour les descriptions, et un modèle PLNE pour sélectionner les meilleurs clusters. L'utilisation des motifs fermés réduit la redondance des clusters candidats par rapport à d'autres méthodes, et le modèle PLNE donne à notre approche exacte plus de flexibilité, et plus de garantie sur les solutions fournies par rapport aux approches heuristiques. Nous voulons exploiter les techniques de recherche locale afin d'améliorer l'étape de résolution.

Remerciements. Ce travail a été partiellement financé par le Programme d'excellence Eiffel. UR/EIFFEL-DOCTORAT 2015/840868H. Ce travail a été en partie soutenu par l'ANR, projet Hybride ANR-11-BS002-002.

Références

[1] D. Aloise, P. Hansen, and L. Liberti. An improved column generation algorithm for minimum sum-of-squares clustering. *Math. Prog.*, 131(1-2) :195–220, 2012.

[2] B. Babaki, T. Guns, and S. Nijssen. Constrained clustering using column generation. In *CPAIOR 2014*, pages 438–454, 2014.

[3] P. Berkhin. A survey of clustering data mining techniques. In *Grouping Multidimensional Data - Recent Advances in Clustering*, pages 25–71. 2006.

[4] T-B-H. Dao, K-C. Duong, and C. Vrain. A declarative framework for constrained clustering. In *ECML PKDD*, volume 8190 of *LNCS*, pages 419–434, 2013.

[5] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2) :139–172, 1987.

[6] B. Ganter and R. Wille. *Formal Concept Analysis : Mathematical Foundations*. Springer-Verlag, 1997.

[7] F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *DS 2004*, pages 278–289, 2004.

[8] T. Guns, S. Nijssen, and L. De Raedt. k-pattern set mining under constraints. *IEEE Trans. Knowl. Data Eng.*, 25(2) :402–418, 2013.

[9] P. Hansen, B. Jaumard, and E. Sanlaville. *New Approaches in Classification and Data Analysis*, chapter Partitioning Problems in Cluster Analysis : A Review of Mathematical Programming Approaches, pages 228–240. 1994.

[10] M. Khiari, P. Boizumault, and B. Crémilleux. Constraint programming for mining n-ary patterns. In *16th CP*, volume 6308 of *LNCS*, pages 552–567, 2010.

[11] J-P. Métivier, P. Boizumault, B. Crémilleux, M. Khiari, and S. Loudni. A constraint language for declarative pattern discovery. In *SAC 2012*, pages 119–125, 2012.

[12] R. S. Michalski. Knowledge acquisition through conceptual clustering : A theoretical framework and an algorithm for partitioning data into conjunctive concepts. *Journal of Policy Analysis and Information Systems*, 4(3) :219–244, 1980.

[13] R. S. Michalski and R. E. Stepp. Learning from observation : Conceptual clustering. In *Machine Learning*, pages 331–363. Springer, 1983.

[14] M. Mueller and S. Kramer. Integer linear programming models for constrained clustering. In *DS 2010*, pages 159–173, 2010.

[15] Andrzej J. Osiadacz. Integer and combinatorial optimization. *International Journal of Adaptive Control and Signal Processing*, 4(4) :333–334, 1990.

[16] R. G. Pensa, C. Robardet, and J-F. Boulicaut. A bi-clustering framework for categorical data. In *PKDD 2005*, pages 643–650, 2005.

[17] M. Perkowitz and O. Etzioni. Adaptive web sites : Conceptual cluster mining. In *IJCAI 99*, pages 264–269, 1999.

[18] R. J. Schalkoff. Pattern recognition. In *Wiley Encyclopedia of Computer Science and Engineering*. 2008.

[19] K. Thompson and P. Langley. Concept formation in structured domains. In *Concept Formation : Knowledge and Experience in Unsupervised Learning*, pages 127–162. 1991.

[20] T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *DS 2004*, pages 16–31, 2004.

La XOR-résolution

Nicolas Prcovic
LSIS - Université d'Aix-Marseille
nicolas.prcovic@lsis.org

Résumé

Nous présentons la XOR-résolution, une généralisation de la résolution standard. Elle part de l'idée qu'à partir de deux clauses CNF contenant deux littéraux opposées $x \vee y \vee C_1$ et $\overline{x} \vee \overline{y} \vee C_2$, la résolution ne permet d'inférer que la tautologie $y \vee \overline{y} \vee C1 \vee C_2$ alors que la sous-formule non tautologique $(x \oplus y) \vee C1 \vee C_2$ (où $\oplus$ est le "ou exclusif") pourrait être produite. En généralisant la règle de résolution, nous obtenons un système plus puissant (ie, capable d'obtenir une réfutation de longueur polynômiale plutôt qu'exponentielle de certaines classes d'instances). Nous montrons aussi comment les méthodes de résolution arborescente de type DPLL ou CDCL peuvent intégrer la XOR-Résolution pour être théoriquement plus puissantes.

Abstract

We present XOR-resolution, a generalization of Resolution. It starts from the idea that from two CNF clauses containing two opposite literals $x \vee y \vee C_1$ and $\overline{x} \vee \overline{y} \vee C_2$, Resolution only allows to infer the tautology $y \vee \overline{y} \vee C_1 \vee C_2$ while the non tautological subformula $(x \oplus y) \vee C_1 \vee C_2$ (where $\oplus$ is the "exclusive or") could be produced. By generalizing the Resolution rule, we obtain a more powerful proof system (ie, able to generate a polynomial, instead of exponential, length refutation of some classes of instances). We also show how the Resolution-based tree search methods DPLL or CDCL can integrate XOR-Resolution to be more powerful.

1 Introduction

Les systèmes de preuve basés sur la résolution (Res) [3] sont utilisés pour permettre de trouver la réfutation d'une formule booléenne supposée insatisfiable ou de montrer qu'il existe un modèle. Les méthodes classiques de recherche arborescente de modèle (DPLL, CDCL) peuvent être interprétées comme une manière d'appliquer une suite de résolutions pour obtenir une solution. Cependant, certaines familles d'instances [4, 13, 1] nécessitent de produire un nombre exponentiel de résolvantes. La résolution étendue [12] ajoute une régle supplémentaire à la résolution. Personne n'a jamais trouvé de classe d'instances nécessitant la production d'un nombre exponentiel de résolvantes avec la résolution étendue. Mais, bien que rendant un système de preuve par résolution plus puissant, la règle d'extension est très difficile à mettre en pratique car le nombre d'extensions possibles est très élevé et il est difficile de savoir quand l'ajout d'une extension sera utile ou dommageable.

Nous présentons ici un système de preuve intermédiaire entre la résolution et la résolution étendue. Il est basé sur la remarque que lorsque la règle de résolution génère une tautologie, donc une clause inutile, il est possible à la place d'inférer une sous-formule non tautologique qui apporte une "information" potentiellement utile.

La règle de résolution permet d'obtenir une clause $C_1 \vee C_2$ à partir des clauses $x \vee C_1$ et $\overline{x} \vee C_2$. Mais si un littéral est dans C_1 et que son opposé est dans C_2 alors on a affaire à une tautologie donc $C_1 \vee C_2$ n'est pas ajoutée à l'ensemble des clauses.

Or, il s'avère que l'on peut inférer logiquement une formule non tautologique à partir de $x \vee C_1$ et $\overline{x} \vee C_2$ en généralisant la règle de résolution. Nous allons le voir en rappelant la raison pour laquelle on peut ajouter la résolvante $C_1 \vee C_2$ à l'ensemble des clauses.

L'ensemble des clauses représente une conjonction de clauses F qu'on peut exprimer ainsi : $F = (x \vee C_1) \wedge (\overline{x} \vee C_2) \wedge F'$.

Comme on a la propriété $A = A \wedge (A \vee B)$, la formule peut se réécrire successivement :
$F = ((x \vee C_1) \wedge (x \vee C_1 \vee C_2)) \wedge ((\overline{x} \vee C_2) \wedge (\overline{x} \vee C_2 \vee C_1)) \wedge F'$
$F = (x \vee C_1) \wedge (\overline{x} \vee C_2) \wedge ((x \wedge \overline{x}) \vee C_1 \vee C_2) \wedge F'$
$F = (x \vee C_1) \wedge (\overline{x} \vee C_2) \wedge (C_1 \vee C_2) \wedge F'$

Or, si nous avons deux clauses $x \vee y \vee C_1$ et

$\overline{x} \vee \overline{y} \vee C_2$, le même type de raisonnement s'applique :
$F = (x \vee y \vee C_1) \wedge (\overline{x} \vee \overline{y} \vee C_2) \wedge F'$
$F = ((x \vee y \vee C_1) \wedge (x \vee y \vee C_1 \vee C_2)) \wedge ((\overline{x} \vee \overline{y} \vee C_2) \wedge (\overline{x} \vee \overline{y} \vee C_2 \vee C_1)) \wedge F'$
$F = (x \vee y \vee C_1) \wedge (\overline{x} \vee \overline{y} \vee C_2) \wedge (((x \vee y) \wedge (\overline{x} \vee \overline{y})) \vee C_1 \vee C_2) \wedge F'$

Or, $(x \vee y) \wedge (\overline{x} \vee \overline{y}) = x \oplus y$, $\oplus$ étant le "ou exclusif". On peut donc ajouter $(x \oplus y) \vee C_1 \vee C_2$ à la formule (plutôt qu'une tautologie inutile).

Bien sûr, si z est dans C_1 et $\overline{z}$ est dans C_2, $(x \oplus y) \vee C_1 \vee C_2$ est une tautologie. Mais nous pouvons généraliser le raisonnement, quelque soit le nombre de littéraux en opposition dans les deux clauses : à partir des clauses $x_1 \vee ... \vee x_n \vee C_1$ et $\overline{x_1} \vee ... \vee \overline{x_n} \vee C_2$, on peut ajouter $((x_1 \vee ... \vee x_n) \wedge (\overline{x_1} \vee ... \vee \overline{x_n})) \vee C_1 \vee C_2$. Or, $x_1 \vee ... \vee x_n$ signifie que tous les x_i ne sont pas égaux à 0 tandis que $\overline{x_1} \vee ... \vee \overline{x_n}$ signifie que tous les x_i ne sont pas égaux à 1. La conjonction des deux équivaut au fait que parmi les x_i, il y en a au moins un qui diffère des autres. On peut donc réexprimer $(x_1 \vee ... \vee x_n) \wedge (\overline{x_1} \vee ... \vee \overline{x_n})$ par $(x_1 \oplus x_2) \vee ... \vee (x_1 \oplus x_n)$ qui signifie que x_1 (on aurait pu choisir n'importe quelle autre variable de manière équivalente) diffère au moins d'un des autres x_i. Au final, la formule inférée est $(x_1 \oplus x_2) \vee ... \vee (x_1 \oplus x_n) \vee C_1 \vee C_2$ qui est une disjonction d'exclusions, non tautologique.

Nous voyons que les formules que nous pouvons inférer ne sont plus des clauses CNF (des disjonctions de littéraux) mais des disjonctions d'exclusions, ce qui nous amène à généraliser la notion de clause afin de définir un système de preuve qui généralise Res.

Dans cet article, nous allons donc définir la notion de x-clauses, de formule XCNF et la XOR-résolution, qui généralisent respectivement les clauses CNF, les formules CNF et la résolution standard. Nous montrerons que la XOR-résolution constitue un système de preuve plus puissant que la résolution mais moins puissant que la résolution étendue. Nous montrerons ensuite comment prendre en compte la XOR-résolution dans les méthodes de recherche arborescente pour rendre les solveurs plus puissant (en théorie).

2 Notions préliminaires

Une *formule booléenne*, sous sa forme dite *CNF*, est une conjonction de clauses. Une clause est une disjonction de littéraux. Un littéral est soit une variable booléenne, soit sa négation. On définit var(l) comme étant la variable du littéral l. Les variables peuvent être affectées à la valeur 1 (vrai) ou 0 (faux). Une clause peut être représentée par l'ensemble de ses littéraux. Une formule SAT peut être représentée par l'ensemble de ses clauses. Un *modèle* d'une formule F est une affectation de variables qui est telle que F est vraie. Une formule n'ayant pas de modèle est dite *insatisfiable*. La clause vide, notée $\square$, est toujours fausse et si une formule la contient alors elle est insatisfiable. Si une clause contient un littéral l et son opposé $\neg l$, elle est toujours vraie et on l'appelle *tautologie* sur var(l).

2.1 Resolution

Les systèmes formels de preuve propositionnelle permettent de dériver d'autres formules à partir d'une formule F, grâce à des règles d'inférence. En particulier, le système de Robinson [10], que nous appelons `Res`, permet de dériver des clauses induites à partir des clauses d'une formule F grâce à la règle de *résolution* :

$$\frac{C_1 \qquad C_2}{C_1 \backslash \{l\} \cup C_2 \backslash \{\bar{l}\}}$$

La clause inférée par résolution de C_1 avec C_2 sera appelé *résolvante* sur var(l) de C_1 et C_2. L'intérêt de `Res` est qu'il permet d'établir des réfutations de formules (ie, des preuves de leur insatisfiabilité) dans la mesure où une formule est insatisfiable si et seulement si il existe une suite de résolutions qui dérive la clause vide. `Res` fonctionne par saturation de l'application de sa règle : si une formule est satisfiable, `Res` le détecte quand plus aucune résolvante non redondante ne peut être dérivée.

Une dérivation est une suite d'applications de la règle de résolution permettant de dériver une clause. Elle constitue la preuve que cette clause est une conséquence logique de la formule. Une dérivation de la clause vide s'appelle une *réfutation*. On appelle *longueur d'une preuve*, le nombre de résolvantes qu'elle contient.

Nous rappelons qu'on dit qu'un système de preuves P *p-simule* un système de preuves Q ssi il existe une fonction reformulant en temps polynomial une preuve produite par P à partir d'une preuve produite par Q. Ainsi, P est dit *plus puissant* que Q si P peut p-simuler Q mais Q ne peut pas p-simuler P. Informellement, un système de preuves propositionnel P est plus puissant que Q si P peut générer des preuves de longueur polynômiale pour des classes d'instances pour lesquelles Q ne peut générer que des preuves de longueur exponentielle, et que lorsque Q peut générer une preuve de longueur polynômiale, P le peut aussi.

2.2 Le "ou exculsif" (XOR)

Nous rappelons succintement quelques propriétés de l'opérateur $\oplus$. Il est commutatif et associatif.

$a \oplus b = a \wedge \overline{b} \vee \overline{a} \wedge b = (a \vee b) \wedge (\overline{a} \vee \overline{b})$.

$a \oplus b$ peut s'interpréter comme "a diffère de b". $a_1 \oplus a_2 \oplus ... \oplus a_n$ peut s'interpréter comme "il y a

un nombre impair de variables a_i qui sont égales à 1 (vraies)" ou comme une équation linéaire dans le corps Z/2Z, c'est-à-dire "la somme modulo 2 des variables a_i égale 1". C'est pourquoi nous les appellerons *équations booléennes*, mais on les retrouve aussi ailleurs sous le nom de *contraintes de parité*.

Voici quelques relations que nous utilisons dans cet article :
$a \oplus 1 = \overline{a}$ (1)
Simplifications :
$a \oplus a = 0$ (2)
$a \oplus b \vee a = a \vee b$ (3)
$a \oplus b \vee \overline{a} = \overline{a} \vee \overline{b}$ (4)
$a \oplus b \wedge a = \overline{b}$ (5)
$a \oplus b \wedge \overline{a} = b$ (6)
Redistribution des variables dans les équations :
$a \oplus b \vee a \oplus c = a \oplus b \vee b \oplus c$ (7)
$a \oplus b \wedge a \oplus c = a \oplus b \wedge b \oplus c \oplus 1$ (8)

En fait, les relations (3), (4), (5) et (6) sont des cas particuliers des relations (7) et (8) pour lesquelles on a $b = 0$ ou $b = 1$.

L'opérateur d'équivalence $\Leftrightarrow$ est la négation du $\oplus$: $x \Leftrightarrow y = \overline{x \oplus y} = x \oplus y \oplus 1$. Toute équation booléenne est reformulable en une chaîne d'équivalences en remplaçant chaque $\oplus$ par un $\Leftrightarrow$ et en ajoutant "$\oplus 1$" si le nombre de $\oplus$ était impair.

On normalise l'expression d'une équation booléenne en utilisant d'abord la relation (1) pour faire disparaître les occurences de littéraux négatifs, puis en utilisant la relation (2) pour faire disparaître les doublons de variable ou de 1. À la fin, une équation booléenne ne contient plus de négation, aucune variable n'apparaît plus d'une fois, et 1 apparaît une fois au plus.

Les systèmes linéaires d'équations booléennes (ie, les conjonctions d'équations booléennes) se résolvent grâce à la méthode du pivot de Gauss. Quand un tel système a au moins une solution, l'ensemble des équations s'exprime d'une manière normalisée en ordonnant arbitrairement les variables. On fait alors en sorte que la première variable n'apparaisse plus à partir de la deuxième équation, que la deuxième variable n'apparaisse plus à partir de la troisième équation, etc. Cela se fait en utilisant un nombre polynômial de fois la relation (8).

Ex : $a \oplus b \oplus c \wedge a \oplus d \oplus e \wedge b \oplus d \oplus 1 =$
$a \oplus b \oplus c \wedge (b \oplus c \oplus 1) \oplus d \oplus e \wedge b \oplus d \oplus 1 =$
$a \oplus b \oplus c \wedge b \oplus c \oplus 1 \oplus d \oplus e \wedge (c \oplus d \oplus e) \oplus d \oplus 1 =$
$a \oplus b \oplus c \wedge b \oplus c \oplus 1 \oplus d \oplus e \wedge c \oplus e \oplus 1.$
À ce moment, on réécrit les équations, toujours en utilsant la relation (8) de manière à ce qu'elles contiennent des variables dont l'ordre est le plus éloigné possible : $b \oplus c \oplus 1 \oplus d \oplus e = b \oplus e \oplus 1 \oplus d \oplus e = b \oplus d \oplus 1$ et $a \oplus b \oplus c = a \oplus d \oplus e$. Au final, la normalisation de la conjonction d'équations donne $a \oplus d \oplus e \wedge b \oplus d \oplus 1 \wedge c \oplus e \oplus 1$.

Lorsqu'on normalise un système d'équations booléennes, on le résout : le système est inconsistent ssi l'équation vide (0) est produite pendant la normalisation. Le problème XORSAT de décision sur l'existence d'une solution d'une conjonction d'équations booléennes est donc polynomial.

2.3 Formules XCNF

On appelle *x-clause* (ou simplement *clause* à partir de maintenant, lorsque le contexte n'est pas ambigü), une disjonction d'équations booléennes. Les x-clauses ont déjà été introduites (sous le nom de *clauses étendues*) dans [6] dans le cadre de la compilation de connaissances dédiée au comptage de modèles. En imitant la normalisation des systèmes d'équations booléennes, on normalise la formulation des x-clauses grâce à la relation (7). Précisément, on ordonne arbitrairement les variables et on applique la relation (7) de manière à ce que la première variable n'apparaisse pas dans les équations booléennes suivantes, que la deuxième variable n'apparaisse plus à partir de la troisième équation, etc. Ensuite, comme pour la normalisation des conjonctions d'équations, on réécrit les équations de manière à ce qu'elles contiennent des variables dont l'ordre est le plus éloigné possible. Cette normalisation permet de diminuer au maximum le nombre d'équations dans la disjonction.
Ex : $a \oplus b \vee a \oplus c \vee b \oplus c =$
$a \oplus b \vee b \oplus c \vee b \oplus c =$
$a \oplus b \vee b \oplus c \vee c \oplus c =$
$a \oplus b \vee b \oplus c$

Elle permet aussi de détecter les tautologies.
Ex : $a \oplus b \vee a \oplus c \vee b \oplus c \oplus 1 =$
$a \oplus b \vee b \oplus c \vee b \oplus c \oplus 1 =$
$a \oplus b \vee b \oplus c \vee c \oplus c \oplus 1 = 1$

Une *formule XCNF* (appelée E-CNF dans [6]) est une conjonction de x-clauses. Toute formule CNF peut se réécrire comme une formule XCNF dans la mesure où il suffit de remplacer chaque littéral négatif $\overline{x}$ par $1 \oplus x$. Une instance XCNF dont toutes les clauses sont unaires (ie, ne contiennent qu'une équation) équivaut à une instance XORSAT (et l'ensemble de ces instances constitue donc une classe polynômiale du problème XCNF).

3 La XOR-résolution

Etant données deux x-clauses, on définit deux règles qui permettent d'en inférer une troisième pour l'ajouter à l'ensemble des x-clauses de la formule. Ces deux

règles constituent le système de preuves que nous appelons XOR-résolution (`XRes`).

$$\text{R1} : \frac{Q_1 \vee ... \vee Q_n \vee C_1 \qquad \overline{Q_1} \vee ... \vee \overline{Q_n} \vee C_2}{Q_1 \oplus Q_2 \vee ... \vee Q_1 \oplus Q_n \vee C_1 \vee C_2}$$

où chaque Q_i est une équation booléenne et $\overline{Q_i}$ désigne son opposée.

$$\text{R2} : \frac{x \oplus Q_1 \vee C_1 \qquad x \oplus Q_2 \vee C_2}{1 \oplus Q_1 \oplus Q_2 \vee C_1 \vee C_2}$$

où x est une variable, et Q_1 et Q_2 sont des équations booléennes. R2 se justifient par la relation $x \oplus Q_1 \wedge x \oplus Q_2 = 1 \oplus Q_1 \oplus Q_2 \wedge x \oplus Q_1$. Cette règle permet d'inférer une x-clause quand deux x-clauses ont chacune une équation booléenne qui possède une variable x commune.

R1 et R2 sont des généralisations de la règle de résolution. R1 équivaut à la règle de résolution si $n = 1$, R2 si $Q_1 = 0$ et $Q_2 = 1$.

La règle R2 permet d'obtenir une x-clause sans x dans la mesure où il est toujours possible de normaliser une x-clause de telle manière que x n'apparaissent qu'une seule fois dans chacune des deux x-clauses. Il suffit que x soit considérée comme la première variable de son ensemble. Cependant, on ne doit appliquer R2 que si elle ne produit pas une tautologie. Sinon, c'est la règle R1 qu'il faut appliquer.

3.1 Puissance de la XOR-résolution

Définition 1 *Soit $X = \{x_1, x_2, ..., x_k\}$. On note $Eq(X)$ l'équation booléenne $x_1 \oplus x_2 \oplus ... \oplus x_k$.*
On note $CNF(X)$ l'ensemble (insatisfiable) de toutes les clauses CNF de longueur $|X|$ que l'on peut faire avec les variables de X.
On note $CNF(X)^{\oplus}$ l'ensemble de toutes les clauses CNF dont la conjonction équivaut à $Eq(X)$.
On note $CNF(X)^{\oplus 1}$ le complémentaire de $CNF(X)^{\oplus}$ par rapport à $CNF(X)$.

Ex : $CNF(\{a,b,c\}) = \{a \vee b \vee c, a \vee b \vee \overline{c}, a \vee \overline{b} \vee c, a \vee \overline{b} \vee \overline{c}, \overline{a} \vee b \vee c, \overline{a} \vee b \vee \overline{c}, \overline{a} \vee \overline{b} \vee c, \overline{a} \vee \overline{b} \vee \overline{c}\}$. $CNF(\{a,b,c\})^{\oplus} = \{a \vee b \vee c, a \vee \overline{b} \vee \overline{c}, \overline{a} \vee b \vee \overline{c}, \overline{a} \vee \overline{b} \vee c\}$. $CNF(\{a,b,c\})^{\oplus 1} = \{a \vee b \vee \overline{c}, a \vee \overline{b} \vee c, \overline{a} \vee b \vee c, \overline{a} \vee \overline{b} \vee \overline{c}\}$. La conjonction des clauses de $CNF(\{a,b,c\})^{\oplus}$ équivaut à $Eq(\{a,b,c\}) = a \oplus b \oplus c$ et celle de $CNF(\{a,b,c\})^{\oplus 1}$ à $Eq(\{a,b,c\}) \oplus 1 = a \oplus b \oplus c \oplus 1$.

Proposition 1 *Soit $X = \{x_1, x_2, ..., x_k\}$. À partir des clauses de $CNF(X)^{\oplus}$, on obtient la x-clause $Eq(X)$ en appliquant $2^{k-1} - 1$ fois la règle R1 de* `XRes`.

On procède en faisant des résolutions avec la règle R1 entre couples de clauses de la forme $C \vee x_i \vee x_{i+1} \oplus ... \oplus x_k$ et $C \vee \overline{x_i} \vee x_{i+1} \oplus ... \oplus x_k \oplus 1$, qui produit $C \vee x_i \oplus x_{i+1} \oplus ... \oplus x_k$, et couples de clauses de la forme $C \vee \overline{x_i} \vee x_{i+1} \oplus ... \oplus x_k$ et $C \vee x_i \vee x_{i+1} \oplus ... \oplus x_k \oplus 1$, qui produit $C \vee x_i \oplus x_{i+1} \oplus ... \oplus x_k \oplus 1$. Au début, $i = n-1$, on a les 2^{k-1} clauses de $CNF(X)^{\oplus}$ et on produit 2^{k-2} résolvantes. Puis, à partir de ces résolvantes, pour $i = k-2$, on produit 2^{k-3} résolvantes selon le même schéma, et on continue pour toutes les valeurs de i décroissante jusque $i = 0$, où on obtient la résolvante $x_1 \oplus x_2 \oplus ... \oplus x_k$. En tout, on a appliqué $2^{k-1} - 1$ fois la règle R1. □

Proposition 2 *La XOR-résolution est strictement plus puissante que la Résolution standard.*

Dans la mesure où les x-clauses sont des généralisations des clauses CNF et que les règles de `XRes` sont des généralisations de la règle de résolution, `XRes` est au moins aussi puissant que `Res`. Nous allons montrer que les problèmes d'Urquhart, qui se réfutent en temps nécessairement exponentiel avec `Res` [13], se réfutent en temps polynomial avec `XRes`. Les problèmes d'Urquhart ne contiennent que des clauses appartenant à des ensembles $CNF(X_i)^{\oplus}$ ou $CNF(X_i)^{\oplus 1}$, où les X_i sont des sous-ensembles non disjoints de l'ensemble de toutes les variables. Par ailleurs, chaque X_i contient sept variables au maximum. À une équation booléenne portant sur k variables correspond 2^{k-1} clauses CNF. Un problème d'Urquhart qui serait formulé sous la forme d'un ensemble d'équations booléennes se résoudrait en temps polynômial grâce par exemple à la méthode du pivot de Gauss, qui se simule par l'application de la règle R2 (avec $C_1 = 0$ et $C_2 = 0$). Or, en partant des clauses d'un ensemble $CNF(X_i)^{\oplus}$ ou $CNF(X_i)^{\oplus 1}$, on peut obtenir chacune de ces équations grâce à $2^7 - 1$ applications de la règle R2 au maximum d'après la proposition 1. Donc toute instance du problème d'Urquhart se résout en temps polynômial. □

3.2 XOR-résolution et résolution étendue

La résolution étendue [12] consiste à ajouter à `Res` la *règle d'extension* en plus de la règle de résolution. Nous appellerons `ER` ce système de preuve. En toute généralité, la règle d'extension permet d'ajouter (à l'ensemble des clauses) les clauses correspondant à la formule $x \Leftrightarrow F$, où x est une nouvelle variable et F est n'importe quelle formule portant sur des variables existant déjà. L'intérêt de rajouter la règle d'extension est qu'elle rend le système plus puissant dans la mesure où certains problèmes dont la preuve est de longueur nécessairement exponentielle dans `Res` ont une preuve polynômiale dans `ER`. C'est le cas du fameux problème des pigeons [4, 2].

Chaque x-clause peut se réécrire comme la conjonction d'une clause CNF et d'équations booléennes :

$Q_1 \vee ... \vee Q_n =$
$(z_1 \vee ... \vee z_n) \wedge (z_1 \Leftrightarrow Q_1) \wedge ... \wedge (z_n \Leftrightarrow Q_n) =$
$(z_1 \vee ... \vee z_n) \wedge (z_1 \oplus Q_1 \oplus 1) \wedge ... \wedge (z_n \oplus Q_n \oplus 1)$. Les z_i sont des variables introduites par la règle d'extension. Ainsi, `XRes` peut être vu comme une façon restreinte d'appliquer la résolution étendue. Ses deux règles se simulent ainsi :

$$\text{R1'} : \frac{x_1 \vee ... \vee x_n \vee C_1 \qquad \overline{x_1} \vee ... \vee \overline{x_n} \vee C_2}{z_1 \vee ... \vee z_{n-1} \vee C_1 \vee C_2,\ z_1 \Leftrightarrow x_1 \oplus x_2,\ ...,\ z_{n-1} \Leftrightarrow x_1 \oplus x_n}$$

$$\text{R2'} : \frac{z_1 \vee C_1 \qquad z_2 \vee C_2 \qquad z_1 \Leftrightarrow x \oplus y_1 \qquad z_2 \Leftrightarrow x \oplus y_2}{z_3 \vee C_1 \vee C_2,\ z_3 \Leftrightarrow 1 \oplus y_1 \oplus y_2}$$

Chaque formule de type $a \Leftrightarrow b \oplus c$ équivaut à la conjonction des quatre clauses CNF $\overline{a} \vee b \vee c$, $a \vee \overline{b} \vee c$, $a \vee b \vee \overline{c}$, $\overline{a} \vee \overline{b} \vee \overline{c}$. On peut donc réécrire les règles R1' et R2' uniquement avec des clauses CNF.

4 XDPLL : la recherche arborescente sur une formule XCNF

Il paraît peu probable que la définition d'un solveur basé directement sur la résolution d'une instance CNF grâce à l'application répétée des règles de la XOR-résolution lui permette d'être compétitif avec les solveurs SAT complets les plus récents, tous basés sur la recherche arborescente. Il nous faut donc trouver comment effectuer une recherche arborescente qui simule la XOR-résolution, au moins partiellement. Dans cette section, nous montrons comment simuler l'application de la règle R2.

Pour résoudre une instance F, un solveur de type DPLL ou CDCL utilise le fait que $F = (F \wedge x) \vee (F \wedge \overline{x})$, où x est une variable apparaissant dans F. Il procède par dichotomie en considérant $F \wedge x$ et $F \wedge \overline{x}$. $F \wedge x$ se déduit de F en retirant les clauses contenant x, car la clause unitaire x les subsume, et en retirant le littéral $\overline{x}$ des clauses qui le contenait, obtenant ainsi les résolvantes entre les clauses contenant $\overline{x}$ et la clause unitaire x (grâce à la propagation unitaire).

On peut généraliser le procédé en remarquant que $F = (F \wedge Q) \vee (F \wedge (Q \oplus 1))$ où Q est une équation booléenne contenant des variables de F (comme cela a déjà été proposé dans [6], mais pour compiler de la connaissance). Ainsi, on remplace le choix de variable par le choix d'un sous-ensemble de variables, et le choix de valeur par le choix d'une des deux équations booléennes sur ces variables. Cela étant, afin de gérer correctement l'adjonction d'une équation booléenne dans la formule correspondant à un sous-arbre de recherche, nous allons devoir généraliser la notion de propagation unitaire.

4.1 Propagation unitaire dans les XCNF

Dans le contexte d'une instance XCNF, une clause unitaire est une clause uniquement constituée d'une équation booléenne. La propagation unitaire va consister à choisir une équation booléenne Q et une variable x de Q et d'appliquer jusqu'à saturation la règle R2 entre la clause unitaire $Q = x \oplus Q_1$ et des clauses $x \oplus Q_2 \vee C_2$. Précisément, on va remplacer toutes les occurences de x par $Q_1 \oplus 1$ dans toutes les autres clauses (ce qui correspond à la suppression de x dans la propagation unitaire classique). Si des clauses unitaires sont produites, on recommence le processus avec une clause unitaire produite en choisissant une nouvelle variable de cette clause, qui disparaîtra à son tour de la formule.

En fait, l'application de R2 entre une clause unitaire Q et une clause C permet aussi de détecter si Q subsume C et donc d'éliminer C de la formule, comme l'indique la proposition suivante.

Proposition 3 *Si l'application de R2 entre une x-clause unitaire Q et une x-clause C produit une tautologie alors Q subsume C.*

Pour que R2 s'applique, il faut qu'il existe une variable x telle que $Q = x \oplus Q_1$ et $C = x \oplus Q_2 \vee C_1$. La résolvante est $Q_1 \oplus Q_2 \oplus 1 \vee C_1$. Pour que cette résolvante soit tautologique, il faut que C_1 soit vraie quand $Q_1 \neq Q_2$. Or, $x \oplus Q_1$ est vraie quand $x \neq Q_1$ et alors :

- soit $x \neq Q_2$ et alors $x \oplus Q_2 \vee C_1$ est vrai.
- soit $x = Q_2$ donc $Q_2 \neq Q_1$ donc C_1 est vrai donc $x \oplus Q_2 \vee C_1$ est vrai.

Donc, quand $x \oplus Q_1$ est vrai, C est vrai donc $x \oplus Q_1$ subsume C. □

Comme on l'a vu, la résolvante se détecte comme étant une tautologie lorsque qu'elle est normalisée.

Nous pouvons maintenant définir XDPLL, une généralisation de DPLL :

```
XDPLL(F) :
  % Entrée : F, une formule XCNF
  % Sortie : vrai ssi F est satisfiable
   Si F est vide
    retourne vrai
   Si F contient la clause vide
      retourne faux
   Si F contient une clause unitaire Q
      x = choisit-variable(Q)
      Q1 = Q[x|0]
      retourne XDPLL(F[x|Q1+1])
   Q = choisit-equation(F)
   x = choisit-variable(Q)
   Q1 = Q[x|0]
   retourne XDPLL(F[x|Q1]) ou XDPLL(F[x|Q1+1])
```

Q[x|0] désigne l'équation où x est supprimé (en le mettant à 0). F[x|Q] désigne la transformation de la formule F où toutes les occurrences de x de F ont été remplacées par Q puis chacune des x-clauses a été renormalisée afin d'être simplifiée voire supprimée si c'est une tautologie.

4.2 Puissance de XDPLL

Proposition 4 *XDPLL est plus puissante que DPLL.*

Nous montrons que les instances d'Urquhart peuvent être aussi résolue en temps polynômial par XDPLL. En effet, une instance d'Urquhart F représente un ensemble d'équations booléennes sur des ensembles de variables X_i, c'est-à-dire une union d'ensembles $CNF(X_i)^{\oplus}$ ou $CNF(X_i)^{\oplus}1$. Lors de la recherche, à chaque point de choix, il suffit de choisir un ensemble X_i de variables de $CNF(X_i)^{\oplus}$ et d'explorer les deux branches de l'arbre de recherche $F \wedge Eq(X_i)$ et $F \wedge Eq(X_i) \oplus 1$. Supposons que F contient les clauses $CNF(X_i)^{\oplus}$ (le cas où ce serait $CNF(X_i)^{\oplus 1}$ est équivalent). Alors :

- Chacune des clauses de $CNF(X_i)^{\oplus}$ est subsumée par $Eq(X_i)$. Cela est détecté par la propagation unitaire. Donc dans le sous-arbre de recherche qui explore l'alternative $F \wedge Eq(X_i)$, on supprime toutes les clauses de $CNF(X_i)^{\oplus}$.
- La sous-formule $CNF(X_i)^{\oplus} \wedge Eq(X_i) \oplus 1$ est insatisfiable et ne contient que sept variables au maximum. Donc le sous-arbre de recherche qui explore l'alternative $F \wedge Eq(X_i) \oplus 1$ mène à un échec qui peut être détecté en temps constant si on choisit systématiquement une (équation constituée d'une seule) variable de X_i.

Globalement, l'arbre de recherche est déterminé par une succession d'alternatives où on se demande si chacune des équations booléennes induites par une instance du problème d'Urquhart est vraie ou fausse. Quand on suppose qu'elle est fausse, cela est réfuté en temps constant. On se retrouve donc à la fin à supposer que chacune des équations est vraie et il ne reste plus qu'à vérifier si le système d'équations est vrai, ce qui se fait par propagation unitaire. Si une instance d'Urquhart représente m équations booléennes, on a vérifié m fois qu'aucune équation n'est fausse dans l'instance puis on a résolu une fois le système d'équations. La résolution du problème s'est donc effectuée en temps polynômial. □

Nous avons montré que XDPLL pouvait être plus efficace en théorie que la recherche arborescente traditionnelle DPLL avec choix d'une seule variable. Cependant, il faut admettre qu'elle est plus difficile à mettre en œuvre que DPLL car il ne s'agit plus seulement de choisir une variable à chaque point de choix mais un sous-ensemble des variables. Cet aspect pratique est traité dans [6] où le choix des sous-ensembles de variables (qui constituent une équation) est basé sur le score VSIDS de leur variable.

5 Travaux connexes

Nous présentons quelques travaux ayant été effectués sur la prise en compte de relations d'équivalence ou de "ou exclusif" dans les instances SAT et les comparons avec notre approche.

Fondamentalement, ces travaux se résument à extraire syntaxiquement ou sémantiquement d'une formule CNF des équations booléennes puis à résoudre l'instance en faisant cohabiter une formule CNF et un ensemble d'équations booléennes. Les détections syntaxiques se font typiquement dans la phase de prétraitement de la formule, avant de lancer le solveur. Les détections sémantiques se font en chaque point de choix.

Le solveur eqSatz décrit dans [8] maintient un ensemble de chaînes d'équivalences de taille maximum 3 qu'il met à jour en chaque nœud de l'arbre de recherche. Il utilise ces chaînes pour faire de la propagation (par exemple par substitution quand une chaîne est de taille 2) et les détecte sémantiquement par propagation unitaire. On peut voir eqSatz comme une forme limitée mais très pragmatique de recherche arborescente à la XDPLL dans la mesure où il ne gère que des chaînes d'équivalence au plus ternaire (ce qui correspondrait à des équations booléennes sur trois variables).

Dans [7] est proposé un solveur qui gère séparément deux ensembles de clauses : des clauses CNF et des équations booléennes. Un solveur SAT s'occupe des clauses CNF et communique avec un autre solveur qui gère un système d'équations booléennes. Les deux solveurs communiquent afin que le filtrage dans l'un ait des répercussions dans l'autre. Les travaux dans [5] utilisent aussi cette technique des deux solveurs mais en permettant une détection sémantique plus poussée d'équations (non limitée en taille) et en indiquant comment obtenir des équations booléennes syntaxiquement pendant le pré-traitement de la formule (fondamentalement : en repérant les ensembles de clauses CNF qui équivalent à une équation booléennes). Dans [9], une approche syntaxique permet non seulement de détecter des chaînes d'équivalences mais aussi d'autres types de fonctions entre variables booléennes.

Si on considère la réécriture R1' et R2' des clauses R1 et R2 dans le cadre de la résolution étendue (cf section 3.2), il suffit de remarquer que $a \Leftrightarrow b \oplus c = a \oplus b \oplus c \oplus 1$, pour que R1' et R2' puissent être considérée comme mettant en jeu uniquement des clauses CNF et

des équations booléennes :

$$\frac{x_1 \vee ... \vee x_n \vee C_1 \qquad \overline{x_1} \vee ... \vee \overline{x_n} \vee C_2}{z_1 \vee ... \vee z_{n-1} \vee C_1 \vee C_2,\ 1 \oplus z_1 \oplus x_1 \oplus x_2,\ ...,\ 1 \oplus z_{n-1} \oplus x_1 \oplus x_n}$$

$$\frac{z_1 \vee C_1 \qquad z_2 \vee C_2 \qquad 1 \oplus z_1 \oplus x \oplus y_1 \qquad 1 \oplus z_2 \oplus x \oplus y_2}{z_3 \vee C_1 \vee C_2,\ z_3 \oplus y_1 \oplus y_2}$$

Les travaux de [7] peuvent donc aussi être interprétés comme une façon de faire partiellement de la XOR-résolution. La différence majeure est que l'ensemble des équations booléennes est donné et ils ne peuvent en produire d'autres qu'en les combinant entre elles après avoir affecté certaines variables pendant la recherche. Les points de choix de la recherche arborescente sont celui d'un choix de variable et pas un choix d'équations comme nous le faisons. En conséquence, cela revient à ne simuler l'application de la règle R2 que dans le cas où $C_1 = C_2 = 0$, celle qui correspond à de la propagation unitaire d'équations.

À note connaissance, les seules techniques d'inférences (syntaxiques) qui ont été proposées se limitent à la découverte d'équations booléennes à partir de clauses CNF et n'atteignent jamais le cas plus général que nous avons présenté où il s'agit d'obtenir une disjonction d'équations booléennes à partir d'autres disjonctions d'équations booléennes.

Dans cet article, nous n'avons pas abordé l'aspect pratique d'un solveur basé sur la XOR-résolution et les difficultés que cela engendre. Ces difficultés ont été abordés dans d'autres articles à travers la gestion des équations booléennes : comment choisir une équation booléenne à chaque point de choix [6], comment gérer efficacement des équations booléennes plutôt que des littéraux [7, 11, 5], etc.

6 Conclusion et perspectives

En définissant la XOR-résolution, nous avons défini un cadre théorique général pour la prise en compte d'équations booléennes (ou de chaînes d'équivalences) sous-jacentes à une formule que son expression CNF empêche de traiter efficacement. En définissant les x-clauses, nous intégrons le ou-exclusif dans les formules à traiter au lieu de maintenir d'un côté un ensemble de clauses CNF et d'un autre côté un ensemble d'équations booléennes, de raisonner sur les deux ensembles séparément et de les faire communiquer. Les deux règles de la XOR-résolution unifient tous les traitements possibles et permettent d'inférer plus de clauses que ce que font les solveurs dédiés jusqu'à présent.

Une fois posé ce cadre, nous avons indiqué comment la recherche arborescente pouvait simuler l'application de la règle R2 de la XOR-résolution, qui généralise la règle de résolution standard et permet donc d'être plus efficace en théorie. Il nous reste à voir comment on pourrait aussi simuler l'application de la règle R1. Une façon d'y parvenir serait d'abord de définir une variante de DP60 [3], un algorithme complet qui applique la règle de résolution jusqu'à trouver la clause vide (problème insatisfiable) ou jusqu'à obtenir la formule vide (problème satisfiable) en éliminant une par une les variables de la formule. Nous pourrions nous en inspirer pour définir une méthode complète permettant d'appliquer R1 et R2 pour décider de la satisfiabilité d'une formule CNF. Cependant, il n'est pas évident de procéder par éliminations successives de variable de la formule. En effet, seule la règle R2 permet de se "débarasser" d'une variable x mais elle n'est pas toujours applicable et c'est alors R1 qu'il faut appliquer, mais toutes les variables subsistent dans la résolvante.

La prochaine étape de nos travaux pourrait donc être de trouver une variante de DP60 pour la XOR-résolution afin d'en déduire une méthode de recherche arborescente qui simule complètement la XOR-résolution, à la façon dont DPLL avait été défini à partir de DP60.

Références

[1] Vasek Chvátal and Endre Szemerédi. Many hard examples for resolution. *J. ACM*, 35(4) :759–768, 1988.

[2] Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8 :28–32, 1976.

[3] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3) :201–215, 1960.

[4] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39 :297–308, 1985.

[5] Marijn Heule and Hans van Maaren. Aligning CNF- and equivalence-reasoning. In *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, pages 145–156, 2004.

[6] Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. Knowledge compilation for model counting : Affine decision trees. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013.

[7] Tero Laitinen, Tommi A. Junttila, and Ilkka Niemelä. Conflict-driven xor-clause learning. In

Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings, pages 383–396, 2012.

[8] Chu Min Li. Integrating equivalency reasoning into davis-putnam procedure. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 291–296, 2000.

[9] Richard Ostrowski, Éric Grégoire, Bertrand Mazure, and Lakhdar Sais. Recovering and exploiting structural knowledge from CNF formulas. In *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, pages 185–199, 2002.

[10] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1) :23–41, 1965.

[11] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, pages 244–257, 2009.

[12] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logics*, pages 115–125, 1968.

[13] Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1) :209–219, 1987.

Réification de Contraintes Tables

Minh Thanh Khong[1]***Christophe Lecoutre**[2] **Yves Deville**[1] **Pierre Schaus**[1]

[1] ICTEAM, Université catholique de Louvain, Belgium
[2] CRIL-CNRS UMR 8188, Université d'Artois, F-62307 Lens, France
{minh.khong,yves.deville,pierre.schaus}@uclouvain.be lecoutre@cril.fr

Résumé

La réification d'une contrainte c consiste à lui associer une variable booléenne b telle que la satisfaction ou non satisfaction de c corresponde à la valeur de vérité de b, ce que l'on peut noter $c^{reif} : c \Leftrightarrow b$. La réification s'avère utile, entre autres, pour combiner logiquement des contraintes et comptabiliser le nombre de contraintes réifiées satisfaites. Étant donné que les contraintes tables jouent un rôle important en programmation par contraintes, dans cet article, nous nous intéressons à leur réification, en proposant notamment un algorithme de filtrage établissant la cohérence d'arc généralisée, sans sur-coût de mémoire. Nous montrons également comment la décomposition de la contrainte `soft-regular` en contraintes tables ternaires réifiées se traduit par une mesure de violation originale.

Abstract

Reifiyng a constraint c consists in associating a Boolean variable b with c such that c is satisfied if and only if b is true, which can be denoted by $c^{reif} : c \Leftrightarrow b$. Reification is useful for logically combining constraints and counting how many reified constraints can be satisfied. Since table constraints play an important role within constraint programming, in this paper, we are interested in their reification. First, we present a filtering algorithm to establish generalized arc consistency on reified table constraints, with no spatial overhead. Next, we show how the decomposition of the soft global constraint `soft-regular` into reified ternary table constraints involves an original violation measure.

1 Introduction

La réification d'une contrainte c est utilisée pour refléter sa valeur de vérité dans une variable booléenne b. Par conséquent, réifier une contrainte c consiste à remplacer c par sa forme réifiée $c^{reif} : c \Leftrightarrow b$, avec maintenant la possibilité que c puisse être violée : b est vrai si et seulement si la contrainte c est satisfaite. Les contraintes réifiées sont utiles pour appliquer des connecteurs logiques entre les contraintes ou exprimer qu'un certain nombre de contraintes doit être satisfait, par exemple, en additionnant les variables (interprétées comme 0-1) associées aux contraintes réifiées [7].

Les contraintes tables, c.-à-d., les contraintes définies en énumérant explicitement les combinaisons acceptées (ou interdites) de valeurs pour les variables de leurs portées, jouent un rôle important en programmation par contraintes. En effet, elles peuvent être considérées comme un service à usage général, offert par les solveurs de contraintes, pour exprimer toute sorte de contraintes, avec la consommation en espace requis comme seule limite à cette approche. Les contraintes tables peuvent être utiles pour combiner efficacement certaines parties de problèmes (par exemple, la combinaison de contraintes fortement liées), et apparaissent naturellement dans de nombreux domaines tels que la configuration de produits et les bases de données. De nombreux algorithmes ont été proposés au fil des ans pour filtrer les contraintes tables [2, 15, 9, 20, 13, 11, 14, 6], ou certaines de leurs formes compactes [12, 4, 10, 18].

Ces dernières années, un certain nombre de travaux ont été proposés pour la réification des contraintes globales [8, 1, 7], ne comprenant toutefois pas les contraintes tables. Dans cet article, nous étudions la réification des contraintes tables, notamment en décrivant un algorithme pour établir la cohérence d'arc généralisée ou *Generalized Arc Consistency* (GAC), suivant la technique de réduction tabulaire simple ou *Simple Tabular reduction* (STR) [20, 13]. Un résultat intéressant de notre travail est que la porte est ouverte à la réification de tous types de contraintes, sous réserve qu'il soit possible en pratique de les reformuler

*Papier doctorant : Minh Thanh Khong[1] est auteur principal.

en contraintes tables.

Nous montrons aussi comment les contraintes tables réifiées peuvent être exploitées dans le cadre de la contrainte `soft-regular`. Les mesures de violation existantes pour `soft-regular` sont basées sur le concept de distance en terme de variables ou d'opérations d'édition [21]. Une mesure de violation naturelle alternative consiste à compter le nombre de fois qu'une transition inexistante dans l'automate de la contrainte doit être utilisée. Cette mesure de violation est directement obtenue après décomposition d'une contrainte `soft-regular` en contraintes tables ternaires réifiées.

Le papier est organisé comme suit. Tout d'abord, la section 2 introduit quelques pré-requis techniques. Ensuite, en section 3, nous présentons un algorithme de filtrage pour les contraintes tables réifiées, et, en section 4, nous montrons comment les contraintes tables réifiées peuvent être utiles pour traiter `soft-regular`. Avant de conclure, quelques résultats expérimentaux préliminaires sont donnés dans la section 5.

2 Pré-requis techniques

2.1 Réseaux de contraintes

Un problème de satisfaction de contraintes (CSP) P, appelé également un réseau de contraintes, est composé d'un ensemble de n variables, $X = \{x_1, \ldots, x_n\}$, et d'un ensemble de e contraintes, $C = \{c_1, \ldots, c_e\}$. Chaque variable x a un domaine associé, noté $dom(x)$, qui contient l'ensemble des valeurs qui peuvent être assignées à x. La taille maximale d'un domaine pour un CSP donné sera notée d. Chaque contrainte c se compose d'un ensemble ordonné de variables, appelé portée (ou scope) de c et noté par $scp(c)$. Chaque contrainte c est définie par une relation, notée $rel(c)$, qui contient les combinaisons autorisées de valeurs pour les variables de $scp(c)$. L'arité d'une contrainte c est la taille de $scp(c)$, et sera généralement désigné par r.

Étant donné une séquence $\langle x_1, \ldots, x_i, \ldots, x_r \rangle$ de r variables, un r-tuple τ pour cette séquence de variables est une séquence de valeurs $\langle a_1, \ldots, a_i, \ldots, a_r \rangle$, où la valeur individuelle a_i est également noté $\tau[x_i]$ ou, lorsqu'il n'y a pas d'ambiguïté, $\tau[i]$. Soit c une contrainte d'arité r, un r-tuple τ est *valide* sur c ssi $\forall x \in scp(c), \tau[x] \in dom(x)$. L'ensemble des tuples valides sur c est $val(c) = \Pi_{x \in scp(c)} dom(x)$. Un tuple τ est *autorisé* par une contrainte c ssi $\tau \in rel(c)$; on dit aussi que c accepte τ. Un *support* sur c est un tuple valide sur c qui est également accepté par c. Une contrainte table positive (resp. négative) est une contrainte dont la sémantique est définie en extension en énumérant l'ensemble des tuples *autorisés* (resp. *interdits*). Cette table (ensemble) est dénoté $table^{init}(c)$.

Une contrainte c est dite *entailed* (resp. *disentailed*) si tous les tuples τ dans $val(c)$ sont acceptés (resp., non acceptés) par c ; en d'autres termes, c est toujours satisfaite (resp. violée).

Une contrainte c est *Generalized Arc-Consistent* (GAC) ssi $\forall x \in scp(c)$, $\forall a \in dom(x)$, il existe un *support* de (x, a) sur c, c.-à-d., un tuple valide τ sur c tel que τ est accepté par c et $\tau[x] = a$.

Une solution pour un CSP est une affectation d'une valeur à chaque variable telle que toutes les contraintes soient satisfaites. Un CSP est dit *cohérent* s'il admet au moins une solution.

2.2 Réification de Contraintes Tables

La réification d'une contrainte c est obtenue en associant une variable booléenne b à c. On obtient alors une contrainte réifiée $c^{reif} : c \Leftrightarrow b$. La sémantique opérationnelle d'un algorithme de filtrage (propagateur) pour la réification d'une telle contrainte est donnée par les règles suivantes :

- si b est à 1, alors c doit être satisfaite,
- si b est à 0, alors c doit être violée,
- si c devient *entailed*, alors b doit être à 1,
- si c devient *disentailed*, alors b doit être à 0.

Pour traiter une contrainte réifiée, nous avons besoin d'un propagateur pour c, un propagateur pour $\neg c$, et nous devons détecter quand c devient entailed ou disentailed. La proposition ci-dessous montre comment on peut déterminer qu'une contrainte table devient *entailed* et *disentailed*.

Proposition 1 *Soit $table(c)$ l'ensemble des supports courants de c, c.-à-d., nous avons $table(c) = table^{init}(c) \cap val(c)$. Nous avons :*

- *c est entailed ssi $|table(c)| = |val(c)|$,*
- *c est disentailed ssi $|table(c)| = 0$.*

Quand une contrainte table devient *entailed*, tous les tuples valides dans c sont des supports de c. L'exemple 1 montre une contrainte qui est *entailed*.

Exemple 1 Étant donné une contrainte table positive c telle que $scp(c) = \{x_1, x_2, x_3\}$ et $table(c)$ défini par :

x_1	x_2	x_3
0	0	0
0	1	0
1	0	0
1	1	0

Si $dom(x_1) = dom(x_2) = \{0, 1\}$ et $dom(x_3) = \{0\}$, alors c est *entailed* parce que $|table(c)| = 4 = |val(c)| = |dom(x_1) \times dom(x_2) \times dom(x_3)|$.

3 Un Algorithme de Filtrage pour GAC

Dans cette section, nous présentons un algorithme de filtrage qui applique GAC sur une contrainte table réifiée donnée. Le principe est de mettre à jour la table de la contrainte réifiée, à chaque appel de l'algorithme de filtrage, de manière à éliminer les tuples qui sont devenus invalides, puis vérifier l'état *entailment* (ou *disentailment*). Pour la gestion de la table, nous utilisons la technique bien connue appelée STR (*Simple Tabular Reduction*) [20, 13]. Ci-dessous, nous présentons les structures de données utilisées par STR, puis nous présentons l'algorithme de filtrage.

3.1 Structures de donnés

La table associée à une contrainte table c est notée $c.table$. Cette table est représentée par une tableau de tuples indexés de 1 à $c.table.length$ qui représente la taille de la table (c.-à-d. le nombre de tuples autorisés). La complexité en espace dans le pire des cas pour cet ensemble est $O(rt)$ où $t = c.table.length$ et r est l'arité de c.

Les algorithmes basés sur STR ont été développés pour le filtrage des contraintes tables. Ils sont parmi les algorithmes les plus efficaces pour les contraintes tables, en particulier parce que leurs structures de données permettent une restauration peu coûteuse lors des retours-arrières. Le principe de STR est de diviser une table en différents ensembles de telle sorte que chaque tuple soit exactement membre d'un ensemble ; ce qui correspond à l'utilisation des *sparse sets* [3, 5]. L'un de ces ensembles contient tous les tuples qui sont couramment valides : les tuples de cet ensemble constituent le contenu de la *table courante*. Pour simplifier, les structures de données liées au backtracking ne sont pas détaillées dans cet article (voir [13]).

Les tableaux suivants donnent accès aux ensembles disjoints de tuples valides et invalides de $c.table$:

- $c.position$ est un tableau de taille $t = c.table.length$ qui fournit un accès indirect aux tuples de $c.table$. À un moment donné, les valeurs de $c.position$ sont une permutation de $\{1, 2, ..., t\}$. Le i^{eme} tuple de c est $c.table[c.position[i]]$. Pour simplifier, ce tuple est noté $\tau(c, i)$.
- $c.limit$ est la position du dernier tuple courant de $c.table$. La table courante de c est exactement composée de $c.limit$ tuples. Les valeurs dans $c.position$ aux indices allant de 1 à $c.limit$ sont les positions des tuples courants de c.

3.2 STR-Reif

Nous décrivons maintenant STR-Reif, un algorithme pour imposer GAC sur une contrainte table réifiée $c^{reif} : c \Leftrightarrow b$. Cet algorithme est une modification de STR2 [13] appliquée aux contraintes tables réifiées. A noter qu'il peut être utilisé pour une contrainte table positive réifiée ($c.positive = true$) mais aussi pour une contrainte table réifiée négative ($c.positive = false$).

Pour une vérification rapide de la validité des tuples, nous réutilisons l'ensemble S^{val} de STR2 [13] : il contient les variables dont le domaine a été réduit depuis l'invocation précédente de STR-Reif. A la fin de l'invocation de STR-Reif, pour chaque variable $x \in scp(c)$, chaque tuple τ tel que $\tau[x] \notin dom(x)$ a été éliminé de la table courante de c. Au prochain appel, s'il n'y a pas eu de retour-arrière et si $dom(x)$ n'a pas changé, alors $\tau[x] \in dom(x)$ pour chaque tuple courant τ de c. Pour cette raison, lors de la vérification de la validité des tuples, seules les variables présentes dans S^{val} sont vérifiées par l'algorithme 2. A noter que nous utilisions $c.lastSize[x]$ (taille de $dom(x)$ la dernière fois que STR-Reif a été appelé) pour construire S^{val} (lignes 6-10 de l'algorithme 1).

En lignes 1-4, STR-Reif vérifie tout d'abord si b est assignée. Si elle est assignée à 1 (resp, 0), nous avons besoin de poster le propagateur pour assurer c (resp, $\neg c$). Dans ce cas, il n'y a plus d'appel au propagateur de c^{reif}. Il existe de nombreux algorithmes pour cette tâche, par exemple, STR2 [13] pour les contraintes tables positives et STR-Ni [16] pour les contraintes tables négatives. Lorsque $dom(b) = \{0, 1\}$, nous devons vérifier l'entailment ou le disentailment de c (lignes 20-26), après avoir mis à jour la table courante (lignes 11-19).

Proposition 2 *STR-Reif établit GAC sur toute contrainte table réifiée.*

Preuve. Lorsque b est assignée, STR-Reif assure GAC sur la contrainte c si b est 1 ou $\neg c$ si b est 0 (en supposant que les propagateurs pour c et $\neg c$ appliquent GAC). Autrement, STR-Reif garantit de filtrer $dom(b)$ lorsque cela est possible, selon la proposition 1.

Proposition 3 *Pour une contrainte table positive, la complexité temporelle dans le pire de cas de STR-Reif est $O(r't' + r''d)$ où $r' = |S^{val}|$ désigne le nombre de variables pour lesquelles les opérations de validité doivent être effectuées, t' désigne la taille de la table courante de c et r'' le nombre de variables avec des domaines non singletons.*

Preuve. Effectuer le teste de validité se fait en $O(r')$ comme on peut le voir dans l'algorithme 2, puisque

Algorithm 1: STR-Reif($c^{reif}(c, b)$)

```
   // On teste d'abord si b est mis à 0 ou 1
 1 if dom(b)={1} then
 2 |  post(c)
 3 else if dom(b)={0} then
 4 |  post(¬c)
 5 else // dom(b) = {0, 1}
 6 |  S^val ← ∅
 7 |  foreach x ∈ scp(c) : |dom(x)| ≠ c.lastSize[x]
   |  do
 8 |  |  S^val ← S^val ∪ {x}
 9 |  |  c.lastSize[x] ← |dom(x)|
10 |  end
11 |  i ← 1
12 |  while i ≤ c.limit do
13 |  |  if c.isValidTuple(S^val, τ(c, i)) then
14 |  |  |  i ← i + 1
15 |  |  else
16 |  |  |  c.swapTuple(i, c.limit)
17 |  |  |  c.limit ← c.limit − 1
18 |  |  end
19 |  end
   |  // Tester entailment ou disentailment
20 |  if c.limit = 0 then
21 |  |  if c.positive then dom(b) ← {0}
22 |  |  else dom(b) ← {1}
23 |  else if c.limit = |val(c)| then
24 |  |  if c.positive then dom(b) ← {1}
25 |  |  else dom(b) ← {0}
26 |  end
27 end
```

seules les variables présentes dans S^{val} sont considérées. Les boucles en lignes 7-10 et 12-19 de l'algorithme 1 sont $O(r), O(r't')$, respectivement. Lorsque la table est positive, STR2 peut être sollicité (lorsque b est à 1), et sa complexité est en $O(r't' + r''d)$. Donc, la complexité temporelle dans le pire de cas de l'algorithme est $O(r't' + r''d)$.

Notez que la complexité en espace dans le pire de cas de STR-Reif est le même que STR2, qui est, $O(n + rt)$ [13].

4 Décomposition de `soft-regular`

Dans cette section, nous considérons la décomposition de la contrainte `soft-regular` en contraintes ternaires positives réifiées, et introduisons dans ce contexte une nouvelle mesure de violation.

Algorithm 2: isValidTuple(S^{val} : variables, τ : tuple) : Boolean

```
1 foreach variable x ∈ S^val do
2 |  if τ[x] ∉ dom(x) then
3 |  |  return false
4 |  end
5 end
6 return true
```

4.1 Contrainte `regular`

La contrainte globale `regular` a été introduite dans [19, 21]. Elle est définie sur une séquence de variables et indique qu'une séquence de valeurs prises par ces variables doivent appartenir à un langage régulier spécifié. Cette contrainte peut être appliquée, par exemple, pour des problèmes de *rostering* ou *car sequencing*.

Avant d'introduire la contrainte `regular`, nous rappelons la définition d'un automate fini déterministe (DFA).

Un DFA est décrit par un 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ où Q est un ensemble d'états, Σ un alphabet, $\delta : Q \times \Sigma \to Q$ une fonction partielle de transition, $q_0 \in Q$ l'état initial, et $F \subseteq Q$ l'ensemble des états finaux (ou acceptants). Etant donné un mot (une séquence de symboles) en entrée, l'automate démarre dans son état initial q_0 et traite les symboles les uns après les autres, appliquant la fonction de transition δ à chaque étape afin de mettre à jour l'état courant. Le mot est accepté ssi le dernier état atteint appartient à l'ensemble des états finaux F. Les mots acceptés par M sont dits appartenir au langage définit par M, noté $L(M)$. Les langages reconnus par des DFAs sont précisément les langages réguliers.

Par exemple, avec le DFA M représenté par la figure 1, on peut observer que le mot *aaabbaa* est accepté par $L(M)$ tandis que le mot *caab* ne l'est pas.

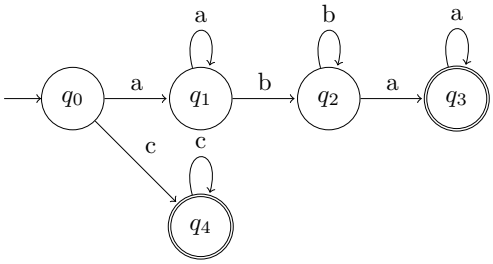


FIGURE 1 – Un DFA avec l'état initial q_0 et les états finals q_3 et q_4.

Définition 1 (Contrainte `regular`) *Soit* $M = (Q, \Sigma, \delta, q_0, F)$ *un DFA et* $X = \langle x_1, \ldots, x_r \rangle$ *une*

séquence de variables. La contrainte $\texttt{regular}(X, M)$ *accepte tous les tuples dans* $\{\langle d_1, \ldots, d_r\rangle \mid d_i \in dom(x_i), \forall i \in 1..r \wedge d_1 \ldots d_r \in L(M)\}$.

Exemple 2 Considérons une séquence de 4 variables $X = \langle x_1, x_2, x_3, x_4\rangle$ avec $dom(x_i) = \{a, b, c\}, \forall i \in 1..4$ et l'automate M représenté par la figure 1. Les tuples (c, c, c, c) et (a, a, b, a) sont acceptés par la contrainte $\texttt{regular}(X, M)$ tandis que (c, a, a, b) et (c, c, a, c) ne le sont pas.

4.2 Décomposition en contraintes tables réifiées

Nous proposons maintenant de décomposer la contrainte `soft-regular` en contraintes ternaires positives réifiées. Cela implique une nouvelle mesure de violation, appelé "dfa", qui est définie comme le nombre de fois où on doit utiliser une transition inexistante dans l'automate de la contrainte. Plus formellement, nous définissons l'ensemble (théorique) des états de longueur $r + 1$ pour un DFA $M = (Q, \Sigma, \delta, q_0, F)$ comme :

$$\mathcal{S}_M^r = \{\langle s_0, \ldots, s_r\rangle \in Q^{r+1} | s_0 \in \{q_0\} \wedge s_r \in F\}$$

Le coût d'un r-tuple τ par rapport à un DFA $M = (Q, \Sigma, \delta, q_0, F)$ et une séquence des états $S = \langle s_0, \ldots, s_r\rangle \in \mathcal{S}_M^r$ est défini comme :

$$\text{dfa-cost}_M(\tau, S) = \#\{i \in 1..r | (s_{i-1}, \tau[i], s_i) \notin \delta\}$$

Le coût d'un r-tuple τ par rapport à un DFA M est :

$$\text{dfa-cost}_M(\tau) = \min_{S \in \mathcal{S}_M^r} \text{dfa-cost}_M(\tau, S)$$

Nous pouvons appliquer cette définition à une contrainte $\texttt{regular}(X, M)$, avec $X = \langle x_1, \ldots, x_r\rangle$, en considérant que le coût de la violation d'un r-tuple τ sur X est $\text{dfa-cost}_M(\tau)$ donné par la définition ci-dessus.

Exemple 3 Lorsque cette nouvelle mesure est utilisée pour la contrainte `regular` de l'exemple 2, nous obtenons $\text{dfa-cost}_M(c, a, a, b) = 2$, en considérant par exemple la séquence des états suivante $\langle q_0, q_4, q_1, q_1, q_3\rangle$.

La contrainte `soft-regular` peut maintenant être définie en considérant la mesure de violation "dfa". Précisément, nous pouvons transformer une contrainte $\texttt{regular}(X, M)$ en une contrainte $\texttt{soft-regular}^{dfa}(X, M, z)$ où z est la variable de coût dont la valeur est définie par la mesure de violation "dfa". Nous pouvons calculer le coût de violation en décomposant la contrainte comme suit. D'abord, nous introduisons $r + 1$ nouvelles variables y_i $(i \in 0..r)$ tel que $dom(y_0) = \{q_0\}$, $dom(y_i) = Q, \forall i \in 1..r - 1$ et $dom(y_r) = F$. Nous introduisons également r variables booléennes z_i, $i \in 1..r$, pour l'objectif de réification. Ensuite, nous introduisons r contraintes tables réifiées $c_i^{reif} : c_i \Leftrightarrow z_i$ où c_i est une contrainte table ternaire positive classique telle que $scp(c_i) = \{y_{i-1}, x_i, y_i\}$ et $rel(c_i) = \delta$ pour $i = 1..r$. Enfin, nous ajoutons une contrainte linéaire $z = \sum_{i \in 1..r}(1 - z_i)$.

Notez que $rel(c_i)$ correspond aux transitions valides dans M, ce qui signifie que $z_i = 1$ ssi $(y_{i-1} x_i y_i) \in \delta$. Par cette propriété, pour tout tuple τ sur X,

$$\text{dfa-cost}_M(\tau) = \min z = \min \sum_{i=1..r} (1 - z_i)$$

Nous devons alors minimiser la valeur de z pour obtenir le coût de violation pour la contrainte `soft-regular`.

Exemple 4 Reconsidérons la contrainte de l'exemple 2. Pour transformer $\texttt{regular}(X, M)$ en $\texttt{soft-regular}^{dfa}(X, M, z)$ avec $X = \langle x_1, x_2, x_3, x_4\rangle$, nous ajoutons les variables y_0, y_1, y_2, y_3 et y_4 ainsi que les variables booléennes z_1, z_2, z_3 et z_4.

Les contraintes tables ternaires réifiées ont la forme $c_i^{reif} : c_i \Leftrightarrow z_i$ avec $rel(c_i)$ définie comme suit (voir les transitions de l'automate en figure 1) :

y_{i-1}	x_i	y_i
q_0	c	q_4
q_4	c	q_4
q_0	a	q_1
q_1	a	q_1
q_1	b	q_2
q_2	b	q_2
q_2	a	q_3
q_3	a	q_3

Le coût de violation "dfa" d'un tuple correspond à la valeur minimale de $(1 - z_1) + (1 - z_2) + (1 - z_3) + (1 - z_4)$, où les valeurs des variables z_i ont été calculées à partir des différentes contraintes tables réifiées. Nous obtenons $z = 2$ pour $\langle c, a, a, b\rangle$ et $z = 1$ pour $\langle c, c, a, c\rangle$.

5 Résultats expérimentaux

Nous avons implémenté STR-Reif dans OscaR [17], un solveur écrit en Scala, et effectué quelques tests sur un Mac OS X 10.10.5 avec a 2.70GHz Intel Core i5 et 16GB de mémoire. Pour tester notre algorithme, nous avons généré des instances du problème académique d'alignement photographique. Dans ce problème, des personnes alignées doivent être prises en photo, et certaines d'entre elles ont des préférences sur les personnes voisines. En fait, pour les instances incohérentes de ce problème, nous avons utilisé des

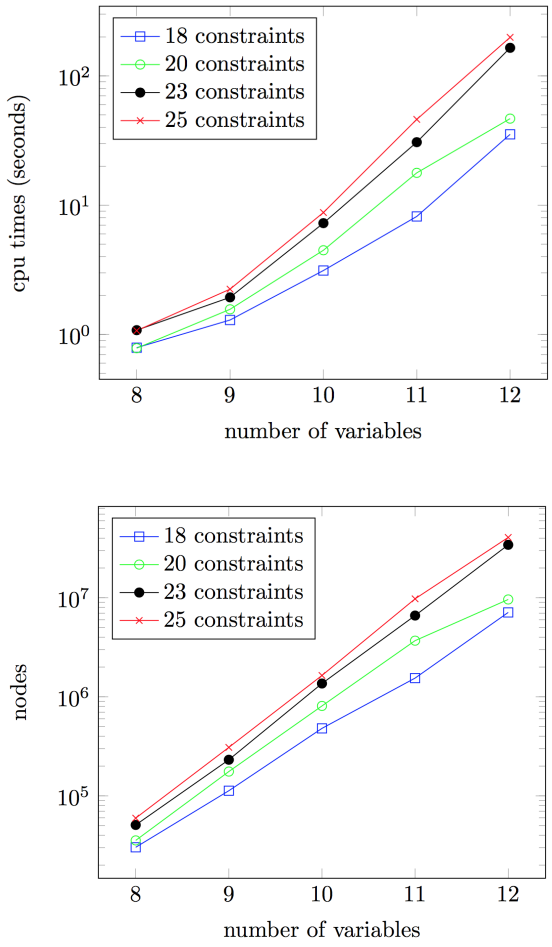


FIGURE 2 – Le temps moyen CPU et le nombre de noeuds visités pour les instances du problème d'alignement photographique

contraintes tables réifiées de manière à maximiser le nombre de contraintes satisfaites (problème MaxCSP).

Pour générer une série d'instances, nous avons utilisé les paramètres d'entrée suivants :

— n, le nombre de variables (c.-à-d. le nombre de personnes),

— e, le nombre de contraintes sur les préférences (c.-à-d. le nombre de préférences entre les personnes).

Dans ce modèle, les variables ont un domaine $1..n$, et il y a une contrainte *alldifferent* pour indiquer que les gens se tiennent à des positions différentes. Nous avons également e contraintes sur les préférences représentées comme des contraintes tables binaires réifiées.

La figure 2 montre le temps moyen cpu (en secondes) et le nombre de noeuds visités, calculés à partir de 10 instances, pour chaque classe de $< n, e >$. On peut observer que l'algorithme monte à l'échelle assez bien (c.-à-d. de manière quasi-linéaire) quand le nombre de variables n ou le nombre de contraintes e augmente. Toutefois, cette expérimentation est tout à fait préliminaire, et nous projetons de faire des expériences plus poussées dans le futur.

6 Conclusion

Dans ce papier, nous avons présenté un algorithme qui permet d'imposer la cohérence d'arc généralisée sur les contraintes tables réifiées sans exiger d'espace supplémentaire : nous utilisons uniquement les tables initiales des contraintes. Nous avons également montré comment il était possible de réifier la version *soft* de la contrainte `regular` en appliquant une simple décomposition en contraintes tables ternaires réifiées, et en adoptant une nouvelle mesure de violation. Comme toute contrainte peut être représentée (en théorie) avec une table, la contrainte table réifiée proposée offre une approche générale pour la réification.

Remerciements Le premier auteur est financé comme Aspirant du FRIA-FNRS belge.

Références

[1] N. Beldiceanu, M. Carlsson, P. Flener, and J. Pearson. On the reification of global constraints. *Constraints*, 18(1) :1–6, 2013.

[2] C. Bessiere and J. Régin. Arc consistency for general constraint networks : preliminary results. In *Proceedings of IJCAI'97*, pages 398–404, 1997.

[3] P. Briggs and L. Torczon. An efficient representation for sparse sets. *ACM Letters on Programming Languages and Systems*, 2(1-4) :59–69, 1993.

[4] K. Cheng and R. Yap. An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15(2) :265–304, 2010.

[5] V. Le clement de saint marcq, P. Schauss, C. Solnon, and C. Lecoutre. Sparse-sets for domain implementation. In *Proceedings of TRICS'13*, 2013.

[6] Jordan Demeulenaere, Renaud Hartert, Christophe Lecoutre, Guillaume Perez, Laurent Perron, Jean-Charles Régin, and Pierre Schaus. Compact-table : Efficiently filtering table constraints with reversible sparse bit-sets. *https ://arxiv.org/abs/1604.06641*.

[7] F. Fages and S. Soliman. Reifying global constraints. Technical Report RR-8084, HAL, 2012.

[8] T. Feydy, Z. Somogyi, and P. Stuckey. Half reification and flattening. In *Proceedings of CP'11*, pages 286–301, 2011.

[9] I.P. Gent, C. Jefferson, I. Miguel, and P. Nightingale. Data structures for generalised arc consistency for extensional constraints. In *Proceedings of AAAI'07*, pages 191–197, 2007.

[10] N. Gharbi, F. Hemery, C. Lecoutre, and O. Roussel. Sliced table constraints : Combining compression and tabular reduction. In *Proceedings of CPAIOR'14*, pages 120–135, 2014.

[11] P. Van Hentenryck J.-B. Mairy and Y. Deville. Optimal and efficient filtering algorithms for table constraints. *Constraints*, 19(1) :77–120, 2014.

[12] G. Katsirelos and T. Walsh. A compression algorithm for large arity extensional constraints. In *Proceedings of CP'07*, pages 379–393, 2007.

[13] C. Lecoutre. STR2 : Optimized simple tabular reduction for table constraints. *Constraints*, 16(4) :341–371, 2011.

[14] C. Lecoutre, C. Likitvivatanavong, and R. Yap. STR3 : A path-optimal filtering algorithm for table constraints. *Artificial Intelligence*, 220 :1–27, 2015.

[15] O. Lhomme and J.-C. Régin. A fast arc consistency algorithm for n-ary constraints. In *Proceedings of AAAI'05*, pages 405–410, 2005.

[16] H. Li, Y. Liang, J. Guo, and Z. Li. Making simple tabular reduction works on negative table constraints. In *Proceedings of AAAI'13*, pages 1629–1630, 2013.

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

[18] G. Perez and J.-C. Régin. Improving GAC-4 for Table and MDD constraints. In *Proceedings of CP'14*, pages 606–621, 2014.

[19] G. Pesant. A regular language membership constraint for finite sequences of variables. In *Proceedings of CP'04*, pages 482–495, 2004.

[20] J.R. Ullmann. Partition search for non-binary constraint satisfaction. *Information Science*, 177 :3639–3678, 2007.

[21] W. van Hoeve, G. Pesant, and L.-M. Rousseau. On global warming : Flow-based soft global constraints. *Journal of Heuristics*, 12(4-5) :347–373, 2006.

Évaluation d'approches complètes pour le problème de somme coloration

Maël Minot[1,2] **Samba Ndojh NDIAYE**[1,3] **Christine SOLNON**[1,2]

[1] Université de Lyon - LIRIS
[2] INSA-Lyon, LIRIS, UMR5205, F-69621, France
[3] Université Lyon 1, LIRIS, UMR5205, F-69622 France
{mael.minot,samba-ndojh.ndiaye,christine.solnon}@liris.cnrs.fr

Résumé

Le problème de somme coloration est un problème de coloration de graphes dans lequel la somme des couleurs utilisées pour chaque sommet doit être minimisée. Peu d'approches complètes ont été proposées à ce jour, et les résultats sont encore largement perfectibles. Cet article évalue les capacités de la programmation par contraintes pour ce problème. Nous étudions différents paramètres et proposons des améliorations visant à mieux adapter la programmation par contraintes à la somme coloration.

Abstract

The sum colouring problem is a graph colouring problem in which the sum of vertex colours must be minimized. Only a few complete approaches were evaluated thus far. This paper evaluates the capabilities of constraint programming to solve this problem. We study several parameters and suggest improvements to better adapt constraint programming to this problem.

1 Introduction

Le problème de somme coloration minimale (MSCP) est une variante $\mathcal{NP}$-difficile du problème de coloration de graphes qui consiste à trouver une coloration minimisant *la somme des couleurs* plutôt que le nombre de couleurs. Ce problème survient notamment dans les contextes d'allocation de ressources et de planification. Le MSCP n'a, à ce jour, pas beaucoup été étudié, et les méthodes proposées sont principalement des métaheuristiques. Seules quelques méthodes complètes ont été avancées : la programmation linéaire entière [18], le *Branch and Bound*, SAT, et la programmation par contraintes (CP) [11]. Cependant, le modèle CP jusqu'alors utilisé est assez direct et n'a pas donné de résultats probants. La programmation par contraintes souples n'a encore jamais été appliquée à ce problème, bien qu'intuitivement elle semble plus indiquée que CP, puisqu'elle est naturellement capable de gérer la préférence entre solutions.

Dans cet article, nous menons une évaluation plus poussée de CP et de sa contrepartie souple, dans le but d'estimer plus justement leur aptitude à résoudre le MSCP. Nous en profitons pour souligner certains avantages de telles approches, ainsi que pour leur apporter plusieurs améliorations dans ce cadre.

2 Définition du MSCP

Un graphe non orienté G est défini par un ensemble de sommets N_G et un ensemble $E_G \subseteq N_G \times N_G$ d'arêtes. Chaque arête est un couple non orienté de sommets. On note $\deg(v)$ le degré d'un sommet v, *i.e.*, $\deg(v) = |\{u \in V, \{u,v\} \in E\}|$, et $\Delta(G)$ le plus grand degré présent dans le graphe G, *i.e.*, $\Delta(G) = \max\{\deg(v), v \in N_G\}$. Une *clique* est un sous-ensemble de sommets tous liés deux à deux. Elle est dite *maximale* si elle n'est pas strictement incluse dans une autre clique. Un *ensemble de sommets indépendants*, a contrario, est un sous-ensemble de sommets au sein duquel on ne trouve aucune arête.

Une k-coloration *propre* ou *légale* de G est une fonction $\mathrm{c} : V \to \{1, \ldots, k\}$ telle que $\forall \{x,y\} \in E, \mathrm{c}(x) \neq \mathrm{c}(y)$. Une telle coloration peut aussi être définie comme une partition de V en k ensembles indépendants.

Le problème de coloration de graphe classique vise à trouver une k-coloration propre qui minimise k. Cet article, cependant, traite du problème $\mathcal{NP}$-difficile de *somme coloration*, dont l'objectif est de trouver une k-coloration propre qui minimise la somme des couleurs

assignées aux sommets ($\sum_{x \in V} c(x)$) plutôt que k. La plus faible somme que l'on peut obtenir pour un graphe G est appelée *somme chromatique de* G et notée $\Sigma(G)$.

Dans [17], des bornes théoriques pour la somme chromatique sont proposées. Elles sont définies de la manière suivante : $\lceil \sqrt{8|E|} \rceil \leq \Sigma(G) \leq \lfloor \frac{3(|E|+1)}{2} \rfloor$. Par la suite, [10] a également démontré qu'une somme coloration optimale n'utilise jamais $\Delta(G)+1$ couleurs. Une autre borne, parfois utile bien que généralement plus faible, est $\Sigma(G) \leq |V| + |E|$.

La *dominance* de certaines solutions par d'autres a été abordée dans [11]. Concrètement, une coloration peut être vue comme une partition ordonnée des sommets du graphe : le k^{e} ensemble S_k correspond aux sommets utilisant la couleur k. Une solution est *dominée* lorsque la somme des couleurs des sommets peut être réduite en réordonnant les indices des ensembles. En d'autres termes, il est alors possible d'améliorer la solution en échangeant des couleurs. Notez qu'une coloration propre le restera lors d'une telle opération. Une partition donnée peut aisément être associée à une coloration que nulle ne domine : il suffit d'utiliser la couleur 1 pour le plus grand ensemble de sommets, la 2 pour le second, et ainsi de suite.

3 Approches existantes pour le MSCP

3.1 Approches incomplètes

De nombreuses approches incomplètes ont été employées pour trouver des solutions approchées au MSCP. La plupart sont passées en revue dans [7], et sont réparties en trois classes : les algorithmes gloutons [19, 20], les heuristiques de recherche locale [2, 6] et les algorithmes évolutionnaires [8, 13, 21].

Notons qu'aucun de ces algorithmes n'est capable d'atteindre toutes les meilleures bornes inférieures ou supérieures. Cependant, le pourcentage de bornes atteintes se situe entre 46 % et 90 %. [21] Ces approches parviennent à prouver l'optimalité d'une solution dès lors que la borne inférieure la plus basse atteint la borne supérieure la plus haute. Cependant, de telles preuves n'ont été possibles que sur un tiers des instances, même en utilisant conjointement les bornes fournies par toutes les méthodes citées par [7].

3.2 Approches complètes

Programmation par contraintes Un problème de satisfaction de contrainte, ou CSP, est défini par un triplet (X, D, C). X est un ensemble fini de variables, D associe un domaine $D(x_i)$ à chaque variable $x_i \in X$, et C est un ensemble de contraintes. Une solution est une affectation de toutes les variables qui satisfait toutes les contraintes.

Un modèle CSP basique pour le MSCP a été proposé dans [11]. Pour un graphe $G = (V, E)$, ce modèle s'obtient en associant une variable x_k à chaque sommet v_k, auxquelles s'applique une contrainte binaire de différence pour chaque arête du graphe :

— $X = \{x_1, \ldots, x_{|V|}\}$
— $\forall i \in \{1, \ldots, |V|\}, D(x_i) = \{1, \ldots, K\}$
— $C = \cup_{\{v_a, v_b\} \in E} \{x_a \neq x_b\}$

où $K = \Delta(G) + 1$. L'objectif est de minimiser $\sum_{x_i \in X} x_i$.

La programmation par contraintes a été évaluée pour le MSCP dans [11], avec le solveur *Choco* [9]. Les résultats étaient assez décevants, avec notamment un temps de résolution de près de 2 000 secondes sur l'instance simple `myciel4`, que les méthodes compétitives résolvent presque instantanément.

Branch and Bound L'approche *Branch and Bound* décrite dans [11] intègre l'idée présentée dans [14, 20], qui consiste à décomposer le graphe en cliques afin de calculer une borne inférieure. Cette technique est appliquée à chaque nœud de l'arbre, en ne considérant que le sous-graphe induit par les sommets qui n'ont pas encore été coloriés. Ainsi est obtenue une borne inférieure globale sur la branche courante de l'arbre. Si cette borne dépasse la borne supérieure courante, cette branche est abandonnée. Pour chaque coloration trouvée, un équivalent dominant est utilisé pour améliorer la borne, comme expliqué à la fin de la section 2. Cette approche donne de meilleurs résultats que le modèle basique CP, mais ne peut être appliquée avec succès qu'à des instances de taille réduite.

SAT Des solveurs SAT ont été employés pour résoudre le MSCP dans [11]. Les résultats rapportés sont très différents de ceux de CP et *Branch and Bound* et sont de qualité inégale : grande efficacité sur des classes d'instances telles que `miles` et `myciel`, mais incapacité à résoudre `queen5_5` en moins d'une heure, malgré une difficulté très raisonnable.

Programmation linéaire Dans [18], un programme linéaire est proposé pour le MSCP. Il associe une variable binaire x_{uk} à chaque paire $(u, k) \in V \times [1, K]$, où u est un sommet, k une couleur et $K = \Delta(G) + 1$, de sorte que $x_{uk} = 1$ si u est colorié avec k, et 0 sinon. L'objectif est de minimiser $f(x) = \sum_{u=1}^{|V|} \sum_{k=1}^{K} k \cdot x_{uk}$ sous les contraintes :

— $c_1 : \sum_{k=1}^{K} x_{uk} = 1, \forall u \in \{1, \ldots, |V|\}$
— $c_2 : x_{uk} + x_{vk} \leq 1, \forall (u, v) \in E, \forall k \in \{1, \ldots, K\}$

La contrainte c_1 force les sommets à n'utiliser qu'une couleur, et c_2 empêche les sommets voisins d'employer la même couleur.

L'article proposant ce modèle rapporte de bons résultats mais également des problèmes dus à un manque de mémoire sur certaines instances. [18]

4 Nouveaux modèles pour le MSCP

Notre but est d'explorer de nouvelles possibilités de modèles pour le MSCP. En section 4.1, nous décrivons un modèle CP souple. La section 4.2 considère la possibilité de remplacer certaines contraintes binaires par des contraintes globales *AllDifferent*. La spécification des domaines initiaux des variables est remise en question dans la section 4.3, tandis que les choix d'heuristiques sont évoqués en 4.4 et 4.5. Une méthode de filtrage est décrite en section 4.6, tandis qu'une technique d'échange de couleurs est présentée en 4.7.

4.1 Modèle WCSP

À ce jour, aucun modèle WCSP n'a été proposé pour le MSCP.

Un WCSP est définit par un triplet (X, D, C) tel que X est un ensemble fini de variables, D une fonction associant à chaque variable $x_i \in X$ son domaine $D(x_i)$, et C un ensemble de fonctions de coût chacune définie sur un sous-ensemble de X. Chaque fonction de coût associe un coût à chaque tuple, un tuple étant une combinaison possible de valeurs pour les variables concernées par la fonction de coût.

Afin de traduire le MSCP en WCSP, les contraintes de différences peuvent être représentées par des fonctions de coût binaires, qui associent un coût infini à tout tuple donnant la même valeur aux deux variables impliquées. Il n'y a généralement pas de fonction objectif explicite dans un WCSP : la somme des fonctions de coûts joue ce rôle. Nous pouvons donc ajouter une fonction de coût unaire par variable, qui associe un coût de n au tuple qui donne la couleur n au sommet associé à cette variable.

Plus formellement, pour un graphe $G = (V, E)$, ce modèle WCSP est défini comme suit, en supposant qu'à chaque variable x_k correspond un sommet k :

- $X = \{x_1, \ldots, x_{|V|}\}$
- $\forall k \in \{1, \ldots, |V|\}, D(x_k) = \{1, \ldots, K\}$
- $C = \{f_1(x_u, x_v), \{u, v\} \in E\} \cup \{f_2(x_u), x_u \in X\}$ avec :
  - $f_1(x_u, x_v) = +\infty$ si $x_u = x_v$, et 0 sinon
  - $f_2(x_u) = x_u$

où $K = \Delta(G) + 1$.

L'objectif est de minimiser la somme des fonctions de coût. Elles garantissent que le coût d'une coloration propre équivaut à la somme des couleurs des sommets et interdisent via un coût infini toute coloration invalide.

4.2 Contraintes globales AllDifferent

Les modèles introduits précédemment utilisent des contraintes de différence binaires pour empêcher les sommets voisins d'utiliser la même couleur. Des modèles légèrement plus élaborés peuvent être obtenus en trouvant des ensembles de contraintes binaires de différence correspondant à des cliques et en les remplaçant par des contraintes globales *AllDifferent*. Il existe de nombreuses manières de sélectionner de telles cliques : elles peuvent être maximales ou non, plus ou moins redondantes, et peuvent être construites par diverses heuristiques. Rechercher la plus grande clique, cependant, n'est pas un choix pertinent, car il s'agit d'un problème $\mathcal{NP}$-difficile.

Nous avons évalué plusieurs méthodes de construction de cliques au cours d'une étude non présentée ici. Les résultats nous ont clairement montré qu'il suffisait de construire de nombreuses cliques de manière gloutonne pour obtenir des résultats difficilement perfectibles. Le choix précis d'heuristiques pour la construction de ces cliques n'a qu'un impact réduit sur les temps de résolution. La méthode de construction que nous avons finalement employée est présentée par l'algorithme 1. Pour chaque sommet v, nous construisons une clique maximale de manière gloutonne (lignes 1–8) : nous partons d'une clique composée uniquement de v et y ajoutons itérativement le candidat de plus haut degré (ligne 5). La procédure se termine par l'élimination des cliques incluses dans d'autres et l'ajout de cliques binaires pour toute contrainte non couverte (lignes 9–12).

La contrainte *AllDifferent* permet une représentation bien plus concise des instances. De plus, un solveur peut l'interpréter comme bon lui semble, en utilisant des contraintes globales spécifiques ou des contraintes binaires, selon son implémentation. Ainsi, nous ne souffrons ici d'aucune perte de généralité.

Notre manière de représenter les *AllDifferent* dans le modèle linéaire est inspirée des contraintes qui y figurent déjà. Nous utilisons simplement des contraintes de la forme $x_{1\,c} + \ldots + x_{k\,c} \leq 1$, en citant, pour chaque couleur c, chaque variable existante associée à un sommet appartenant à la clique considérée.

4.3 Réduction des domaines initiaux

Dans toutes les méthodes complètes présentées, les couleurs qui peuvent être assignées à n'importe quel sommet sont bornées par $\Delta(G) + 1$. Nous proposons d'abaisser cette borne en exploitant le fait que, dans

Algorithme 1 : Constructions de cliques.

Data : Graphe $G = (V, E)$
Result : Ensemble S de cliques de G tel que $\forall \{x, y\} \in E, \exists C \in S$ tel que $\{x, y\} \subseteq C$

```
foreach v ∈ V do
    C ← {v}
    Candidates ← voisins de v dans G
    while Candidates ≠ ∅ do
        x ← Sommet de Candidates avec le plus grand degré dans G
        Ajouter x à C
        Retirer de Candidates les non-voisins de x
    Ajouter C à S
Retirer de S toute clique incluse dans une autre
foreach {x, y} ∈ E do
    if ∄C' ∈ S tel que x, y ∈ C' then
        Ajouter {x, y} à S
```

une solution optimale, la couleur du sommet v est toujours inférieure ou égale à $\deg(v) + 1$.

Propriété 1 *Pour toute somme coloration optimale* c *d'un graphe* $G = (V, E)$, $c(v) \leq \deg(v) + 1, \forall v \in V$.

Pour démontrer cette propriété, supposons qu'elle n'est pas vraie pour une coloration optimale donnée c. Il s'ensuit qu'il existe un sommet v de V pour lequel $c(v) > \deg(v) + 1$. Dans de telles conditions, il existe une valeur x non utilisée dans $\{1, \ldots, \deg(v) + 1\}$, puisque v n'a que $\deg(v)$ voisins. Par conséquent, une meilleure coloration que c peut être obtenue en utilisant la couleur x pour v au lieu de $c(x)$. Donc, c n'est pas optimal, ce qui contredit l'hypothèse de départ.

Il s'agit en réalité d'un cas plus spécifique de la limite fixée aux domaines dans [10]. Ici, nous considérons les sommets *indépendamment*. Nous opérons cette réduction dès la génération des instances. Pour chaque sommet, au lieu d'autoriser toutes les couleurs de $\{1, \ldots, \Delta(G) + 1\}$, nous limitons le domaine de la variable associée au sommet v à $\{1, \ldots, \deg(v) + 1\}$. La différence de taille de domaine peut être conséquente, notamment dans des graphes peu denses. Cette modification des instances peut aisément être appliquée à tous les modèles mentionnés dans cet article.

Sur les 126 instances considérées (cf. section 5.2), la réduction des domaines moyenne est de 46 %. Sur quelques instances, la réduction est nulle, tandis qu'une réduction maximale de 95 % peut être observée sur une instance (`3-Insertions_5`). Des réductions de cette ampleur sont constatées dès lors qu'un petit nombre de sommets ont un très haut degré tandis que les autres ont un degré faible.

4.4 Heuristique de choix de valeur

Étant donné que le but est de minimiser la somme des variables, nous avons opté pour le choix systématique de la plus petite valeur disponible dans le domaine de la variable considérée.

4.5 Heuristiques de choix de variable

Dès lors que l'on évalue des heuristiques pour un problème d'optimisation, un problème de polyvalence survient : il est souhaitable de trouver rapidement une bonne solution afin d'être en mesure de couper des branches de l'arbre de recherche, mais les heuristiques de choix de variable qui excellent dans cette tâche sont généralement moins performantes pour effectuer des preuves d'optimalité. Pour cette raison, un utilisateur pourra porter son choix sur différentes heuristiques selon ce qu'il tente d'accomplir. Nous avons entrepris d'élargir nos expériences afin d'évaluer sept heuristiques de choix de variable :

minDom choisit la variable au plus petit domaine courant ;

minDom/wDeg choisit la variable qui minimise le rapport entre la taille de son domaine courant et son degré pondéré [3] ;

activity choisit la variable dont le domaine a le plus été réduit lors des propagations de contraintes [12] ;

minElim choisit la variable qui possède la plus petite valeur dans son domaine courant, et départage les ex-aequo en choisissant celle pour laquelle la valeur choisie sera retirée du plus petit nombre de domaines de variables voisines ;

dynDeg choisit la variable qui a le plus de voisins non colorés, et départage les ex-aequo en faveur de la variable ayant la plus petite valeur dans son domaine courant, puis de celle ayant le plus petit domaine courant (notez que *minElim* et *dynDeg* tentent de sélectionner une variable ayant au moins une faible valeur dans son domaine, afin de permettre à l'heuristique de choix de valeur d'utiliser cette valeur) ;

minRemove choisit la variable pour laquelle la valeur choisie sera retirée du moins de domaines possible, et départage les ex-aequo selon les règles de *dynDeg* ; *minRemove* maintient, pour chaque variable x_i, les voisins qui ont encore dans leur domaine la valeur qui serait utilisée si x_i était choisie ;

maxDegSizeVal choisit la variable qui maximise $degree \div (size\,of\,domain \times smallest\,value)$; les égalités sont brisées de la même manière qu'avec *dynDeg*.

Quelle que soit l'heuristique, les égalités persistantes sont brisées aléatoirement. Dans la section 5, nous comparons ces heuristiques sur deux critères : la capacité à trouver rapidement une bonne solution, et celle à faire des preuves d'optimalité.

4.6 Filtrage

Nous avons adapté l'algorithme de filtrage utilisé avec le *Branch and Bound* de [11]. Cette méthode construit une partition de cliques sur l'ensemble des sommets non coloriés et déduit une borne inférieure de cette partition. Dans [11], une telle partition est construite à chaque nœud de l'arbre de recherche, afin d'augmenter la précision des bornes résultantes. Dans des expériences préliminaires, cependant, nous avons constaté que ces constructions répétées avaient un coût non négligeable, qui, au bout du compte, dégradaient les performances. En réaction à cela, nous avons décidé de ne calculer une partition de cliques qu'une seule fois, au démarrage de notre programme. Cette partition est construite comme indiqué dans l'algorithme 2.

Algorithme 2 : Construction d'une partition de cliques.

```
Data : Graphe G = (V, E)
Result : Partition P de cliques de G telle que
         ∀v ∈ V, ∃!C ∈ P telle que v ∈ C
1  V_unused ← V
2  while V_unused ≠ ∅ do
3      v ← sommet de V_unused avec le plus haut
       degré dans le graphe induit par V_unused
4      C ← {v}
5      Candidates ← voisins de v dans G
6      while Candidates ≠ ∅ do
7          x ← sommet de Candidates avec le plus
           haut degré dans G
8          Ajouter x à C
9          Retirer de Candidates tout non-voisin de x
10     Ajouter C à P
11     V_unused ← V_unused \ C
```

Pour déduire une borne inférieure d'une partition de cliques, il faut d'abord trouver une borne pour chaque clique qui la compose. Pour une clique C de taille k, cette borne correspond à la somme des k plus petites valeurs présentes dans l'union des domaines des variables de C. L'idée sous-jacente est que dans le meilleur des cas il sera possible d'utiliser ces valeurs. Notez qu'il est nécessaire de choisir k valeurs *différentes* puisque, dans une clique, toute paire de variables est soumise à une contrainte de différence. La solution optimale réelle implique généralement des valeurs plus élevées du fait de la présence de contraintes de différence *entre* les cliques, qui sont ici ignorées.

Dans [11], la partition est calculée sur l'ensemble des sommets non colorés. Dans cet article, en revanche, puisque la partition est calculée au début de la recherche, les sommets n'ont pas encore de couleur. Nous tirons cependant profit des sommets déjà colorés au moment du calcul de la borne puisque nous calculons l'union des domaines *courants* lors du choix des k plus petites valeurs. De ce fait, tout sommet coloré – et même simplement tout domaine réduit – contribue à rendre la borne calculée plus précise.

Tenter trop tôt de couper des branches à l'aide d'une telle méthode de filtrage cause bien souvent une perte d'efficacité : lorsque seuls quelques sommets sont colorés, nous ne disposons pas d'assez d'informations pour estimer la qualité des colorations à venir. Concrètement, la borne alors calculée est de piètre qualité. Afin d'éviter des calculs inutiles, nous fixons une limite empêchant le déclenchement de cet algorithme de filtrage. Nous calculons la somme de toutes les valeurs des variables déjà affectées. Si la distance entre cette somme et et la borne supérieure courante équivaut à plus de 20 % de cette dernière, nous estimons trop peu probable que la borne inférieure que donnerait l'algorithme de filtrage atteigne la borne supérieure, et refusons d'utiliser cet algorithme.

En plus de cette limite visant à ne pas *commencer trop tôt* à utiliser ce filtrage, nous utilisons une limite pour ne pas l'employer *trop tard*, au sens de la profondeur dans l'arbre de recherche. En effet, lorsque seuls peu de sommets restent à colorer, il peut être plus rapide de finir l'exploration de la branche courante de l'arbre de recherche plutôt que de passer du temps à calculer des bornes avec l'algorithme de filtrage. Des expériences qui ne sont pas détaillées ici nous ont permis de constater qu'il est convenable de se retenir d'utiliser ce filtrage lorsque cinq sommets ou moins ne sont pas colorés.

4.7 Échange de couleurs

Comme relevé par [11], certaines solutions peuvent être *dominées* : on peut les améliorer en se contentant d'échanger des couleurs, comme rappelé en section 2. Dans les présents travaux, nous utilisons notamment cette méthode pour abaisser plus rapidement la borne supérieure, afin d'accélérer la résolution. Ainsi, toute solution dominée produite lors de la recherche sera transformée en solution non dominée. Par ailleurs, l'ajout d'une contrainte imposant de trouver une solution meilleure que la précédente permet de ne plus produire une solution dominée par celle-ci.

Il est à noter que la technique d'échange de couleurs a un coût computationnel exceptionnellement bas : le surcoût n'opère que lorsqu'une coloration complète du graphe est trouvée, et les vérifications sur la taille des ensembles représentés par les différentes couleurs sont très simples à effectuer.

5 Évaluation expérimentale

5.1 Procédure expérimentale

Nous avons exécuté les programmes évalués sur un processeur Intel® Xeon® E5-2670 0 à 2,60 Ghz, disposant de 20 480 KB de mémoire cache et de 4 GB de RAM. La réduction des domaines décrite en section 4.3 a été utilisée pour chaque modèle et solveur.

5.2 Benchmark

Nous considérons 126 instances couramment utilisées pour le MSCP [18, 7]. Certaines ont été ajoutées par COLOR02/03/04 [1], mais la plupart sont simplement les instances DIMACS conçues pour le problème de coloration classique [2]. Pour certaines de ces instances, la littérature fournit des bornes inférieures et supérieures [18, 7]. Pour les autres, lorsque nous utilisons une borne supérieure initiale, il s'agit de la meilleure trouvée lors d'expériences précédentes.

Afin d'économiser du temps de calcul pour conduire de plus larges expérimentations, nous avons choisi d'effectuer certaines d'entre elles sur un ensemble réduit d'instances. Lorsque le but était de rapidement trouver une bonne solution avec une limite de 30 minutes et sans borne initiale, nous avons opéré sur une sélection d'instances de toutes les classes, en nous concentrant sur celles qui n'étaient pas triviales et en privilégiant celles où la borne supérieure connue était très haute, afin de pouvoir observer de grandes différences entre les bornes trouvées par les différentes méthodes. Pour les expériences portant sur les preuves d'optimalité, avec 24 heures et une borne supérieure initiale donnée, les résultats sont montrés sur les instances pour lesquelles au moins une méthode a pu améliorer la borne ou prouver son optimalité. Les dernières expériences, ainsi que celles visant à comparer les modèles, ont été menées sur le *benchmark* complet.

Lorsque des contraintes *AllDifferent* sont utilisées, les cliques sont construites en dehors des expériences, lors de la création des instances.

5.3 Implémentation

Nous avons implémenté le modèle CSP dans Gecode 4.2.1 [16]. Le codage de ce modèle est direct et ne sera pas détaillé ici. Le moteur de recherche *Branch and Bound* (BAB) a été sélectionné, et les contraintes *AllDifferent* ont été représentées avec la contrainte « `distinct` » de Gecode. Le niveau de cohérence est GAC (« `ICL_DOM` »).

1. `http://mat.gsia.cmu.edu/COLOR02`
2. `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/`

Tableau 1 – Un résumé des résultats obtenus sans et avec contraintes *AllDifferent*, en utilisant Toulbar, CPLEX et Gecode. Les trois premières lignes donnent, respectivement, les distances minimale, moyenne et maximale entre la meilleure solution trouvée par la méthode et la meilleure solution connue, en pourcentages. « Preuves » (resp. « Pb mem. ») correspond au nombre d'instances pour lesquelles l'optimalité a été prouvée (resp. pour lesquelles l'exécution s'est terminée prématurément par manque de mémoire). « Meill. UB » donne le nombre d'instances sur lesquelles la meilleure solution connue a été atteinte, et « Nb meill. » le nombre de fois où la méthode a été la meilleure sur une instance. La « meilleure » méthode est désignée par le statut (preuve > pas de preuve), puis par la qualité de la solution, et enfin par le temps nécessaire pour la trouver.

		Toulbar		CPLEX		Gecode	
		Binaire	AllDiff	Binaire	AllDiff	Binaire	AllDiff
Dist	Min	0	0	0	0	0	0
	Moy	174,3	465,5	70,2	68,7	**9,2**	**9,2**
	Max	1 143,6	2 863,3	1 119,5	1 119,5	**78,5**	**78,5**
Preuves		19	15	**65**	64	10	10
Pb Mém.		42	97	41	43	**0**	**0**
Meill. UB		22	18	71	**73**	31	32
Nb meill.		5	4	41	**49**	13	22

Nous avons implémenté le modèle WCSP avec Toulbar2 0.9.7.0-R [1]. Toulbar a principalement été conçu pour résoudre des WCSP et utilise des algorithmes de consistance souple de haut niveau (EDAC par défaut [5]) [15, 1].

Pour la programmation linéaire, nous avons utilisé ILOG CPLEX 12.6.2 [4]. ILOG CPLEX traite des programmes linéaires, ce qui impose une représentation purement arithmétique des contraintes de différence, qu'elles soient binaires ou globales.

5.4 Comparaison de Gecode, Toulbar et CPLEX

Le tableau 1 résume les résultats obtenus avec les trois solveurs considérés. Chacun a été évalué sur un modèle sans contraintes *AllDifferent* (« Binaire ») et sur un modèle les incluant (« AllDiff »).

Un problème flagrant dans ces premiers tests est que CPLEX, de même que Toulbar, viennent souvent à manquer de mémoire. Les *AllDifferent* n'ont pas un impact significatif sur cet aspect pour CPLEX. En revanche, ils augmentent nettement la consommation de mémoire pour la version de Toulbar utilisée.

Si on se limite aux instances pour lesquelles il n'y a pas eu de problème de mémoire, on constate que CPLEX est très efficace, et résout le plus grand nombre d'instances (65) tout en trouvant la meilleure solution connue pour 73 d'entre elles. Ses besoins en

mémoire sont donc le principal facteur limitant ; il arrive même que CPLEX soit incapable de fournir la moindre solution intermédiaire. Il en résulte que les solutions trouvées par CPLEX sont en moyenne plus grande de 70 % que les meilleures solutions connues.

Grâce à son haut niveau de filtrage, Toulbar fait presque deux fois plus de preuves que Gecode. Cependant, comme pour CPLEX, la quantité de mémoire utilisée devient problématique sur les instances les plus grandes. En effet, Toulbar utilise principalement des contraintes données en extension, ce qui explique au moins partiellement cette consommation de mémoire. Utiliser des contraintes *AllDifferent* souples induit tant de problèmes de mémoires que cette méthode en devient presque inutilisable dans notre contexte.

Gecode n'est capable de résoudre que dix instances, mais ses besoins en mémoire sont raisonnables, ce qui lui permet de ne jamais en manquer. Il en résulte des bornes plus fines, avec une distance moyenne aux meilleures bornes connues de seulement 9,2 %, tandis que CPLEX et Toulbar obtiennent des moyennes respectives de 68,7 % et 174,3 %. Utiliser des contraintes *AllDifferent* permet à Gecode d'atteindre la meilleure solution connue une fois de plus qu'avec le modèle binaire. Ces contraintes aident Gecode à filtrer, et donc à trouver rapidement des solutions intéressantes.

Puisque CPLEX et Toulbar manquent souvent de mémoire, nous nous concentrerons sur Gecode pour la suite de cet article. Nous considérerons le modèle utilisant les contraintes *AllDifferent*, du fait des résultats sensiblement meilleurs qu'il permet d'obtenir.

5.5 Heuristiques de choix de variable

Le tableau 2 présente les résultats d'une comparaison d'heuristiques pour Gecode, sur 30 minutes sans bornes, tandis que le tableau 3 contient les résultats pour 24 h avec une borne supérieure initiale donnée.

Ces expériences montrent que, contrairement à ce que notre intuition tendrait à nous faire penser, *minDom/wDeg* n'est pas extrêmement efficace, que ce soit sur les tests de 30 minutes ou sur ceux de 24 heures. Une autre heuristique couramment utilisée, *activity*, souffre de difficultés similaires. Les meilleures heuristiques sont *minElim* (pour ce qui est de la recherche rapide d'une bonne solution) et *dynDeg* (pour les preuves). Ces deux heuristiques seront donc celles considérées dans la suite de l'article.

5.6 Intégration du filtrage, de restarts et d'échanges de couleurs

La plupart des heuristiques que nous avons utilisées sont plutôt efficaces quand il s'agit de trouver une bonne première solution, mais peinent à s'éloigner de cette première solution dans l'espace de recherche. Un manque de diversité critique en résulte, et la qualité des solutions finales s'en ressent. Ce phénomène est particulièrement présent avec *minElim*.

Afin d'aider Gecode à trouver des solutions plus variées, nous pouvons l'employer avec des *restarts*. Les restarts ont leur propre ensemble de paramètres ; nous avons effectué une série d'expériences afin de trouver des valeurs adéquates pour ces paramètres. Notre choix s'est porté sur des politiques déjà définies dans Gecode : les restarts géométriques et la politique Luby, associée à une valeur d'échelle de 500.

Afin d'évaluer les différentes améliorations proposées, nous les avons ajoutées une à une et avons mené deux séries d'expériences. La première s'est déroulée avec une limite de 30 minutes et sans borne initiale (tableau 4) ; pour la seconde, une limite de 24 heures était imposée, et une borne supérieure initiale était donnée (tableau 5). Ces bornes viennent de la littérature lorsqu'elles existent, et de nos expériences précédentes dans le cas contraire.

Le filtrage apparaît comme désavantageux pour trouver une bonne solution, mais aide Gecode à faire des preuves d'optimalité. Le paramétrage sélectionné, qui consiste à commencer à utiliser le filtrage lorsque la solution courante est à 20 % de la borne supérieure et à cesser de l'employer quand seules cinq variables ou moins n'ont pas encore de valeur affectée, semble être un compromis intéressant. Il facilite les preuves tout en gardant le surcoût en termes de calculs bas.

Comme prévu, l'utilisation de restarts permet à Gecode de trouver de meilleures bornes, grâce à une plus grande diversité dans les solutions formées. On notera cependant qu'utiliser des restarts dans le cadre des preuves d'optimalité n'est pas un bon choix.

Les échanges de couleurs ayant un coût négligeable, ils ne perturbent jamais le processus de résolution, et parviennent à accélérer la baisse de la borne supérieure sur plusieurs instances.

5.7 Évaluation complète

Nous avons évalué l'impact de ces améliorations sur des exécutions de 24 heures. Les premiers tests ont été effectués avec une borne supérieure initiale, sur les 92 instances pour lesquelles une telle borne est fournie par la littérature ; les tests suivants, sans borne initiale, utilisent le *benchmark* entier.

Le tableau 6 montre que les améliorations proposées sont bénéfiques pour la quasi-intégralité du *benchmark*. Une preuve supplémentaire a pu être faite, et la distance moyenne est presque divisée par deux.

Neuf preuves ont pu être effectuées par *dynDeg* en employant toutes les améliorations. Puisque les restarts ne sont pas adaptés aux preuves d'optimalité,

Tableau 2 – Comparaison d'heuristiques pour Gecode, sur 30 minutes, sans borne initiale. Les bornes supérieures « UB_L » sont celles de la littérature, et « UB » celles obtenues en fin d'exécution, avec « * » en cas de preuve. « t_{UB} » sont les secondes nécessaires à Gecode pour trouver la solution associée à « UB ».

		minDom/wDeg		*minDom*		*dynDeg*		*minRemove*		*minElim*		*activity*		*maxDegSizeVal*	
Instance	UB_L	UB	t_{UB}	UB	t_{UB}	UB	t_{UB}	UB	t_{UB}	UB	t_{UB}	UB	t_{UB}	UB	t_{UB}
DSJC500.5	10 886	15913	133	15736	1694	16261	731	13289	6	**13059**	**1349**	15125	898	15995	1747
DSJR500.1	2 156	2510	4	2477	59	2712	0	2339	1	2331	0	**2279**	**36**	2617	37
flat1000_50_0	25 500	56294	88	54578	34	55567	1119	**45217**	**592**	45519	141	55235	1452	56051	361
4-FullIns_5		12062	113	9231	221	8385	255	6707	739	**6680**	**222**	11506	392	8389	221
games120	443	466	0	467	0	497	0	**451**	**0**	456	0	474	0	492	658
ash958GPIA		4794	220	4753	1044	5180	323	**4324**	**46**	4345	16	4580	10	5474	356
3-Insertions_5		2875	309	2721	997	3855	141	2289	8	**2289**	**7**	2841	3	3808	375
latin_square_10	41 444	49885	813	48955	91	58561	301	46847	56	**46847**	**48**	48934	353	59136	1629
le450_5a	1 350	2337	632	2244	0	2313	416	1863	0	**1752**	**1**	2188	483	2312	41
mug88_25	178	182	189	183	0	185	0	181	195	**180**	**0**	181	0	185	0
myciel7	381	499	547	438	1100	722	33	381	0	**381**	**0**	382	0	625	38
qg.order30	13 950	14283	1701	13996	468	14324	477	13953	2	**13950***	**1**	13981	1044	14327	858
r1000.1		9040	86	8383	37	9009	2	7803	115	7703	2	**7579**	**334**	8964	3
fpsol2.i.3	1 636	1959	1	1695	1	8135	1	1677	882	**1670**	**119**	1809	1	7738	0
inithx.i.3	1 986	2518	1	2033	1	13122	1	2034	2	**2019**	**1**	2053	1	13085	1
mulsol.i.5	1 160	**1163**	**120**	1165	0	3734	0	1165	0	1166	0	1164	0	3694	0
zeroin.i.3	998	1018	11	**999**	**0**	3192	301	1080	0	1034	0	1047	0	3129	54
school1_nsh	2 392	4842	845	4749	181	4900	64	3958	352	**3880**	**818**	4478	1426	4821	156
david	237	257	0	254	1436	372	0	255	11	**245**	**727**	257	0	333	3
homer	1 150	1229	10	1228	581	1392	0	1240	1	**1219**	**17**	1227	0	1301	221
miles500	705	861	229	806	0	864	1154	835	0	761	0	**751**	**0**	824	253
queen11_11	733	886	599	837	35	934	7	821	11	**812**	**1701**	853	1310	943	762
wap06a	13 773	16872	1	16791	575	20282	883	**15374**	**3**	15427	818	15882	1102	16902	1708

Tableau 3 – Comparaison d'heuristiques pour Gecode, sur 24 heures, avec borne supérieure initiale donnée. Les bornes supérieures « UB_L » sont celles de la littérature, et « UB » celles obtenues en fin d'exécution. « t_{UB} » est nombre de secondes nécessitées par Gecode pour trouver la solution associée à « UB ».

		minDom/wDeg			*minDom*			*dynDeg*			*minRemove*			*minElim*			*activity*			*maxDegSizeVal*		
Instance	UB_L	UB	t_{UB}	t	UB	t_{UB}	t	UB	t_{UB}	t	UB	t_{UB}	t	UB	t_{UB}	t	UB	t_{UB}	t	UB	t_{UB}	t
1-FullIns_5		554	0	T	521	2006	T	554	0	T	505	0	T	**499**	**0**	**T**	554	0	T	530	1479	T
2-FullIns_3		93*	0	35964	93*	0	29227	**93***	**0**	**47**	93*	0	49505	93*	0	3819	93*	0	618	93*	0	520
2-FullIns_4		417	0	T	417	0	T	417	0	T	372	2272	T	**363**	**105**	**T**	417	0	T	416	9	T
3-FullIns_3		151	0	T	**150**	**42452**	**T**	151	0	T	151	0	T	151	0	T	151	0	T	151	0	T
3-FullIns_4		735	0	T	735	0	T	735	0	T	**683**	**0**	**T**	683	0	T	735	0	T	722	63886	T
4-FullIns_3		241	0	T	239	0	T	**205**	**328**	**T**	220	55	T	213	37	T	241	0	T	241	0	T
ash331GPIA		1695	8	T	1575	3	T	1736	0	T	**1487**	**1**	**T**	1511	309	T	1608	306	T	1736	0	T
1-Ins._4		119	0	T	119	0	T	**119***	**2997**	**5685**	119	0	T	119	0	T	125	0	T	125	0	T
2-Ins._3	62	62*	0	19	62*	0	14	**62***	**0**	**1**	62*	0	24	62*	0	9	62*	0	13	62*	0	1
3-Ins._3	92	92*	0	29466	92*	0	31770	**92***	**0**	**120**	92*	0	42607	92*	0	4206	92*	0	20847	92*	0	158
4-Ins._3		127	21	T	127	0	T	136	53627	T	127	0	T	**127**	**0**	**T**	130	1	T	136	49990	T
myciel5	93	93	0	T	93	0	T	93*	0	26910	93	0	T	93*	0	29980	**93***	**0**	**2689**	93*	0	4384
qg.order30	13 950	13950*	0	1	13950*	0	1	13950*	0	0	13950*	0	1	**13950***	**0**	**0**	13950*	0	0	13950*	0	0
queen5_5	75	75*	0	964	75*	0	917	75*	0	63	75*	0	20	75*	0	13	75*	0	122	**75***	**0**	**12**

Tableau 6 – La colonne « Simple » présente les résultats d'une version de *minElim* sans filtrage, ni restarts, ni échanges de couleurs. La version dite « Améliorée » inclut quant à elle toutes ces améliorations.

		Simple	Améliorée
Dist	Min	0	0
	Moy	9	5,2
	Max	78,5	72
Preuves		10	11
Meill. UB		30	57
Nb meill.		16	118

nous avons tenté de les retirer, ce qui a permis une preuve supplémentaire et accéléré les preuves existantes (tableau 7).

6 Conclusion et travaux futurs

Dans cet article, nous avons considéré le problème de la somme coloration minimale, et démontré que même si la programmation par contraintes semble à première vue désavantagée, elle possède quelques atouts, comme notamment sa consommation nettement réduite de mémoire, comparée à celle des solveurs de WCSP ou de programmation linéaire. De ce fait, un solveur tel que Gecode peut être appliqué à des instances de très grande taille, et pourra passer plus de temps à trouver des solutions de meilleure qualité. Les compétences d'un tel solveur peuvent être améliorées de plusieurs manières, comme en utilisant des restarts convenablement paramétrés, en employant avec modération un algorithme de filtrage, et en veillant à améliorer les solu-

TABLEAU 4 – Ajout progressif des améliorations présentées, pour la recherche d'une bonne solution en 30 minutes, sans borne initiale. Les bornes présentées sur la gauche sont celles de l'état de l'art. « f20-5 » signifie que le filtrage à base de partitions de cliques a été utilisé, avec les paramètres mentionnés en section 5.6 ; « rl500 » indique l'utilisation de restarts Luby avec une échelle de 500 ; « éch. » signifie que les échanges de couleurs étaient autorisés pour améliorer les solutions trouvées. Les « * » mettent les preuves en évidence.

			minElim				*minElim* f20-5				*minElim* f20-5 rl500				*minElim* f20-5 rl500 éch.			
Instance	LB_L	UB_L	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$
DSJC500.5	2 923	10 886	13059	20	1349	T	13068	20	296	T	12798	17.6	1029	T	**12734**	**17**	**392**	**T**
DSJR500.1	2 069	2 156	2331	8.1	0	T	2331	8.1	0	T	**2287**	**6.1**	**85**	**T**	2287	6.1	92	T
flat1000_50_0	6 601	25 500	45519	78.5	141	T	45520	78.5	53	T	44587	74.9	1280	T	**44243**	**73.5**	**365**	**T**
4-FullIns_5			6680	0	222	T	6680	0	225	T	**6680**	**0**	**205**	**T**	6680	0	238	T
games120	443	443	456	2.9	0	T	456	2.9	0	T	**443**	**0**	**1189**	**T**	444	0.2	722	T
ash958GPIA			4345	2	16	T	4345	2	18	T	4267	0.2	470	T	**4259**	**0**	**513**	**T**
3-Insertions_5			2289	0	7	T	2289	0	6	T	**2289**	**0**	**6**	**T**	2289	0	7	T
latin_square_10	40 950	41 444	46847	13	48	T	46847	13	41	T	45533	9.9	918	T	**45372**	**9.5**	**895**	**T**
le450_5a	1 193	1 350	1752	29.8	1	T	1752	29.8	17	T	1496	10.8	1611	T	**1424**	**5.5**	**1090**	**T**
mug88_25	162	178	180	1.1	0	T	180	1.1	0	T	179	0.6	4	T	**179**	**0.6**	**4**	**T**
myciel7	286	381	**381**	**0**	**0**	**T**	381	0	0	T	**381**	**0**	**0**	**T**	381	0	0	T
qg.order30	13 950	13 950	13950*	0	1	1	13950*	0	1	1	13950*	0	1	1	**13950***	**0**	**1**	**1**
r1000.1			7703	1.4	2	T	7703	1.4	2	T	7607	0.1	801	T	**7597**	**0**	**1359**	**T**
fpsol2.i.3	1 636	1 636	1670	2.1	119	T	1670	2.1	170	T	**1641**	**0.3**	**473**	**T**	1641	0.3	521	T
inithx.i.3	1 986	1 986	2019	1.7	1	T	2019	1.7	1	T	1991	0.3	874	T	**1986**	**0**	**584**	**T**
mulsol.i.5	1 160	1 160	1166	0.5	0	T	1166	0.5	0	T	1164	0.3	3	T	**1164**	**0.3**	**2**	**T**
zeroin.i.3	998	998	1034	3.6	0	T	1033	3.5	35	T	1019	2.1	551	T	**1012**	**1.4**	**0**	**T**
school1_nsh	2 176	2 392	3880	62.2	818	T	3883	62.3	154	T	3361	40.5	714	T	**3149**	**31.6**	**1127**	**T**
david	237	237	245	3.4	727	T	244	3	103	T	**240**	**1.3**	**812**	**T**	240	1.3	867	T
homer	1 129	1 150	1219	6	17	T	1219	6	88	T	1191	3.6	1043	T	**1191**	**3.6**	**932**	**T**
miles500	705	705	761	7.9	0	T	752	6.7	0	T	**727**	**3.1**	**1155**	**T**	730	3.5	869	T
queen11_11	726	733	812	10.8	1701	T	818	11.6	345	T	775	5.7	774	T	**772**	**5.3**	**1654**	**T**
wap06a	12 454	13 773	15427	12	818	T	15427	12	661	T	15098	9.6	581	T	**15041**	**9.2**	**1101**	**T**

TABLEAU 7 – « UB_L » sont les bornes supérieures de la littérature données à Gecode comme bornes initiales. « Toutes » indique que toutes les améliorations ont été employées. « rNon » est la même méthode, mais sans restarts. Si la preuve a pu être effectuée en 24 h, nous donnons le nombre de secondes nécessaires.

Instance	UB_L	Toutes	rNon
2-Insertions_3	62	**0**	1
3-Insertions_3	92	129	**38**
myciel3	21	**0**	**0**
myciel4	45	**0**	**0**
myciel5	93		**41 279**
qg.order30	13 950	0	**0**
qg.order40	32 800	2	**1**
qg.order60	109 800	7	**6**
queen5_5	75	17	**10**
queen8_12	624	**0**	**0**

tions via des échanges de couleurs lorsque cela s'avère possible.

Du fait de l'existence de solutions dominées, comme expliqué en section 5.3, nous avons affaire à de nombreuses redondances dans la procédure de recherche, et ce quelque soit le solveur. Ces redondances peuvent être supprimées en exprimant le problème sous la forme d'une construction de partition de sommets, mais quelques expériences nous ont montré qu'il était difficile d'employer une telle méthode sans pénaliser le solveur, car il devient alors difficile d'avoir une évaluation précise de la qualité d'une solution en cours de construction. Par ailleurs, il faut noter que les échanges de couleurs, combinés avec la contrainte imposant de trouver à chaque fois une meilleure solution, permet déjà d'éviter de multiples énumérations de solutions dominées.

Il sera important d'inclure à l'avenir des modèles SAT dans cette étude. Dans [11], ces modèles ont donné de très bon résultats sur certaines instances (classes `miles` et `myciel`), mais se montraient très insuffisant pour des instances raisonnables telles que `queen5_5`.

N. B. : Résultats détaillés dans la version étendue : `https://hal.archives-ouvertes.fr/hal-01309350`

Ces travaux ont été soutenus par l'ANR SoLStiCe (ANR-13-BS02-0002-01).

Références

[1] D Allouche, S de Givry, and T Schiex. Toulbar2, an open source exact cost function network solver. Technical report, Technical report, INRIA, 2010.

[2] U. Benlic and J.-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] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *ECAI*, volume 16, page 146, 2004.

[4] I. CPLEX. High-performance software for mathematical programming and optimization, 2005.

[5] S. De Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency: Getting closer

TABLEAU 5 – Ajout progressif des améliorations présentées, pour la preuve d'optimalité avec une borne supérieure initiale donnée. Les bornes présentées sur la gauche sont celles de l'état de l'art. « f20-5 » signifie que le filtrage à base de partitions de cliques a été utilisé, avec les paramètres mentionnés en section 5.6 ; « rl500 » indique l'utilisation de restarts Luby avec une échelle de 500 ; « éch. » signifie que les échanges de couleurs étaient autorisés pour améliorer les solutions trouvées. Les « * » mettent les preuves en évidence.

			dynDeg				*dynDeg* f20-5				*dynDeg* f20-5 rl500				*dynDeg* f20-5 rl500 éch.			
Instance	LB_L	UB_L	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$
1-FullIns_5			554	11	0	T	554	11	0	T	499	0	1	T	**499**	**0**	**1**	**T**
2-FullIns_3			93*	0	0	47	93*	0	0	9	**93***	**0**	**0**	**8**	93*	0	0	10
2-FullIns_4			417	13	0	T	417	13	0	T	369	0	2025	T	**369**	**0**	**953**	**T**
3-FullIns_3			151	4.1	0	T	**145***	**0**	**15083**	**29781**	145*	0	85	61805	145*	0	84	54774
3-FullIns_4			735	4.4	0	T	735	4.4	0	T	704	0	14346	T	**704**	**0**	**13963**	**T**
4-FullIns_3			205	0	328	T	205	0	315	T	**205**	**0**	**1**	**T**	205	0	2	T
5-FullIns_3			294	5	0	T	294	5	0	T	287	2.5	54992	T	**280**	**0**	**0**	**T**
ash331GPIA			1736	4.7	0	T	1736	4.7	0	T	**1658**	**0**	**8131**	**T**	1668	0.6	881	T
1-Insertions_4			**119***	**0**	**2997**	**5685**	119*	0	3579	6291	123	3.4	56	T	123	3.4	63	T
2-Insertions_3	55	62	62*	0	0	1	62*	0	0	0	62*	0	0	0	**62***	**0**	**0**	**0**
3-Insertions_3	84	92	92*	0	0	120	**92***	**0**	**0**	**50**	92*	0	0	137	92*	0	0	96
4-Insertions_3			136	7.1	53627	T	**127***	**0**	**14804**	**19122**	127	0	60472	T	127	0	3362	T
myciel5	93	93	**93***	**0**	**0**	**26910**	93	0	0	T	93	0	0	T	93	0	0	T
qg.order30	13 950	13 950	**13950***	**0**	**0**	**0**	13950*	0	0	1	13950*	0	0	1	13950*	0	0	0
r125.1			257	0	1	T	**257***	**0**	**0**	**21069**	257	0	0	T	257	0	1	T
queen5_5	75	75	75*	0	0	63	**75***	**0**	**0**	**15**	75*	0	0	17	75*	0	0	16

to full arc consistency in weighted csps. In *IJCAI*, volume 5, pages 84–89, 2005.

[6] A. Helmar and M. Chiarandini. A local search heuristic for chromatic sum. In *Proceedings of the 9th metaheuristics international conference*, volume 1101, pages 161–170, 2011.

[7] Y. Jin, J.-P. Hamiez, and J.-K. Hao. Algorithms for the minimum sum coloring problem: a review. *arXiv preprint arXiv:1505.00449*, 2015.

[8] Y. Jin, J.-Kao Hao, and J.-Philippe Hamiez. A memetic algorithm for the minimum sum coloring problem. *Computers & Operations Research*, 43:318–327, 2014.

[9] N. Jussien, G. Rochart, and X. Lorca. Choco: an open source java constraint programming library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10, 2008.

[10] M. Kubale. *Graph colorings*, volume 352. American Mathematical Soc., 2004.

[11] C. Lecat, C. Min Li, C. Lucet, and Y. Li. Comparaison de méthodes de résolution pour le problème de somme coloration. In *JFPC'15: Journées Francophones de Programmation par Contraintes*, 2015.

[12] L. Michel and P. Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 228–243. Springer, 2012.

[13] A Moukrim, K Sghiouer, C Lucet, and Y Li. Upper and lower bounds for the minimum sum coloring problem, submitted for publication.

[14] A. Moukrim, K. Sghiouer, C. Lucet, and Y. Li. Lower bounds for the minimal sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36:663–670, 2010.

[15] M Sanchez, S Bouveret, S De Givry, et al. Max-csp competition 2008: toulbar2 solver description. *Proceedings of the Third International CSP Solver Competition*, pages 63–70, 2008.

[16] G. Team. Gecode: Generic constraint development environment, 2006, 2008.

[17] C. Thomassen, P. Erdös, Y. Alavi, P. J Malde, and A. J Schwenk. Tight bounds on the chromatic sum of a connected graph. *Journal of Graph Theory*, 13(3):353–357, 1989.

[18] Y. Wang, J.-Kao Hao, F. Glover, and Z. Lü. Solving the minimum sum coloring problem via binary quadratic programming. *arXiv preprint arXiv:1304.5876*, 2013.

[19] Q. Wu and J.-Kao Hao. An effective heuristic algorithm for sum coloring of graphs. *Computers & Operations Research*, 39(7):1593–1600, 2012.

[20] Q. Wu and J.-Kao Hao. Improved lower bounds for sum coloring via clique decomposition. *arXiv preprint arXiv:1303.6761*, 2013.

[21] J.-K. Hao Y. Jin. Hybrid evolutionary search for the minimum sum coloring problem of graphs, accepted to information sciences feb 2016, 2015.

AMPHAROS : un solveur SAT parallèle adaptatif

Gilles Audemard Jean-Marie Lagniez Nicolas Szczepanski Sébastien Tabary

Univ. Lille-Nord de France. CRIL/CNRS UMR 8188, Lens

{audemard,lagniez,szczepanski,tabary}@cril.fr

Résumé

Nous présentons et évaluons AMPHAROS, un nouveau solveur SAT parallèle fondé sur le paradigme « diviser pour mieux régner ». Développé dans un cadre massivement parallèle, ce solveur lance ses *workers* sur des sous-problèmes représentés par des cubes. En plus du partage classique des clauses apprises, un nouveau type d'information associé aux cubes est échangé. Par ailleurs, nous proposons un nouveau critère afin d'adapter dynamiquement la quantité de clauses partagées et le nombre de cubes. Comme souligné dans la partie expérimentale, ce nouveau schéma de parallélisation est performant et ouvre de nombreuses perspectives.

Abstract

We present and evaluate AMPHAROS, a new parallel SAT solver based on the divide and conquer paradigm. This solver, designed to work on a great number of cores, runs workers on sub-formulas restricted to cubes. In addition to classical clause sharing, it also exchange extra information associated to the cubes. Furthermore, we propose a new criterion to dynamically adapt both the amount of shared clauses and the number of cubes. Experiments show that, in general, AMPHAROS correctly adjusts its strategy to the nature of the problem. Thus, we show that our new parallel approach works well and opens a broad range of possibilities to boost parallel SAT solver performances.

1 Introduction

Les travaux concernant le problème SAT commencent habituellement par rappeler les énormes progrès sur les problèmes de type industriel. En effet, les derniers résultats sont très impressionnants et touchent un grand nombre de problèmes *e.g.* planification, vérification formelle et cryptographie qui sont souvent résolus au moyen d'une réduction vers SAT plutôt qu'avec des solveurs natifs. Néanmoins, si l'on joue l'avocat du diable, nous pouvons observer que cette évolution s'amenuise sensiblement. Effectivement, il devient clairement de plus en plus difficile d'améliorer les solveurs SAT modernes. En outre, le problème SAT souffre de son succès, puisque les formules deviennent de plus en plus compliquées à résoudre (14 instances sur 100 de la SAT RACE 2015 du *Parallel Track* sont encore indéterminées).

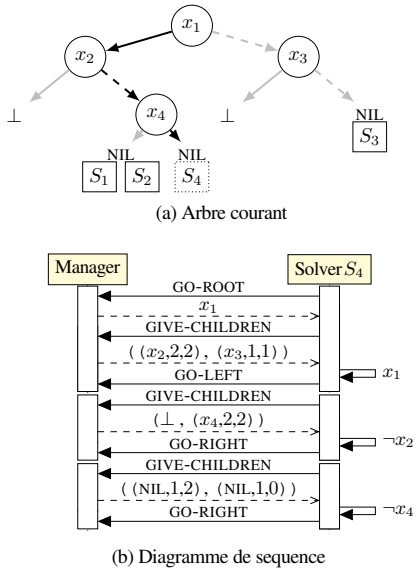
Parallèlement, grâce à l'essor du *cloud computing*, il est désormais possible de solliciter un nombre de machines virtuelles illimitées pouvant être disponibles en quelques secondes. Cependant, comme il a été montré lors de la dernière compétition [32], les solveurs SAT parallèles ne passent pas à l'échelle. En effet, le gagnant du *Parallel Track* a choisi d'utiliser seulement la moitié des cœurs disponibles. Par conséquent, il faut dorénavant concevoir des solveurs parallèles basés sur des nouvelles architectures afin de bénéficier des nombreuses unités de calcul et du *cloud computing*.

Dans la résolution parallèle du problème SAT, les solveurs peuvent être divisés en deux catégories. Premièrement, les solveurs de type *portfolio* lancent, sur la formule originale, différentes heuristiques/stratégies en concurrence [1, 7, 11, 21, 22, 31]. Pendant la résolution, les processus échangent des informations (généralement sous la forme de clauses apprises) afin de s'entraider [1, 6, 21, 22]. Deuxièmement, les solveurs basés sur le paradigme « diviser pour mieux régner » découpent l'espace de recherche en plusieurs sous-espaces afin de les distribuer à des solveurs SAT, communément nommés *workers*. En général, à chaque fois qu'un solveur a fini de résoudre son sous-problème (tandis que les autres travaillent encore), une stratégie d'équilibrage est réalisée afin de transférer dynamiquement un nouveau sous-espace à ce *worker* inactif [13, 14]. Les sous-espaces peuvent être définis en utilisant le concept du chemin de guidage (*guiding path* [37]), généré statiquement (avant la recherche [23, 34]), ou dynamiquement (pendant la recherche [2, 24, 36]). Comme dans les solveurs de type *portfolio*, les clauses apprises peuvent aussi être partagées [16].

Même si la plupart des gagnants du *Parallel Track* de la dernière compétition sont du type *portfolio*, les solveurs de type « diviser pour mieux régner » deviennent de plus en plus efficaces (le solveur TREENGELING a été second [10]). C'est dans ce contexte que nous proposons AMPHAROS, un nouveau solveur SAT parallèle. Cet article constitue la première brique

afin d'atteindre un objectif à long terme : déployer un solveur SAT dans le cloud. Dans notre approche, la formule est partitionnée en utilisant des cubes (comme dans [36]). Un processus appelé MANAGER gère les cubes et chaque solveur travaille sur la formule complète en induisant un de ces cubes. Par opposition aux autres solveurs « diviser pour mieux régner », notre approche peut faire travailler plusieurs solveurs sur le même sous-problème et ces derniers peuvent arrêter la recherche avant d'avoir trouvé une solution ou une contradiction. Cela permet d'éviter qu'un solveur ne reste trop longtemps sur un même sous-problème trop difficile pour lui. Dans ce cas, le solveur demande au MANAGER un autre sous-problème qui peut provenir soit d'un autre cube existant, soit d'une subdivision du sous-problème trop difficile. Notre approche a pour originalité de laisser les solveurs sélectionner eux-mêmes les cubes qu'ils doivent résoudre. De plus, deux sortes de clauses sont échangées : les classiques clauses apprises et d'autres dépendant du cube.

Notre but étant de résoudre le problème SAT dans un cadre massivement parallèle, il est important de proposer une architecture qui adapte sa stratégie suivant le nombre de *workers* et la nature du problème. À cette fin, nous proposons une approche qui utilise un algorithme adaptatif afin d'ajuster simultanément et dynamiquement le nombre de clauses partagées et le nombre de nouveaux cubes (sous-problèmes disponibles). Cela est possible grâce à l'utilisation d'une nouvelle mesure estimant si le processus de recherche doit être intensifié ou diversifié. Nous montrons que quand l'espace de recherche a besoin d'être diversifié (resp. intensifié), la mesure proposée détecte que le nombre de cubes doit être augmenté (resp. diminué) et le nombre des clauses partagées doit être augmenté (resp. diminué).



(a) Arbre courant

(b) Diagramme de sequence

FIGURE 1 – Vue d'ensemble des messages échangés entre le solveurs S_4 et le MANAGER (Fig 1b) suivant une division de cubes sous forme d'arbre (Fig 1a) (une ligne pleine (resp. en pointillés) affectes la variable à vrai (resp. faux).

2 Définitions préliminaires

En raison du manque d'espace, nous supposons les lecteurs habitués avec la logique propositionnelle et à la résolution du problème SAT. Nous rappelons juste quelques aspects des solveurs SAT modernes (CDCL) [28, 35].

La résolution CDCL est un processus de recherche via des branchements, où, à chaque étape, un littéral est sélectionné comme branche. Usuellement, la variable est choisie suivant l'heuristique VSIDS [28] et sa valeur est prise dans un tableau appelé *phase-saving* contenant les précédentes valeurs de toutes les variables assignées par le solveur [30]. Ensuite, la propagation (*Boolean constraint propagation*) est effectuée et peut affecter d'autres variables. Quand un littéral et son opposé sont propagés, un conflit est atteint, une clause provenant de ce conflit est apprise et un *backjump* est réalisé. Ces opérations sont répétées jusqu'à l'obtention d'une solution (satisfaisable) ou qu'une clause vide est produite (insatisfaisable).

Les solveurs SAT CDCL sont souvent enrichis d'une politique de *restart* [18] et d'une stratégie de nettoyage de la base des clauses apprises [3, 5, 17]. Beaucoup d'heuristiques ont été proposées afin d'identifier les clauses les plus pertinentes, la mesure LBD (pour *literal blocked distance*) [5] est l'une des plus efficaces. Le LBD d'une clause apprise correspond au nombre de niveaux distincts impliqués dans celle-ci. De plus, comme le montre les résultats expérimentaux de [5], les clauses obtenant un petit LBD doivent être considérées comme plus pertinentes.

Dans certaines applications, il est nécessaire de résoudre plusieurs instances similaires, et donc de faire, d'une manière incrémentale, plusieurs appels à un solveur SAT [8, 15]. Pour garder une trace des clauses apprises entre ces appels et ainsi, améliorer leurs efficacités, ces solveurs ont la particularité d'utiliser la notion d'hypothèse. Un ensemble d'hypothèse est défini comme un ensemble de littéraux qui va être supposé vrai [15]. Cet ensemble peut être vu comme un cube, i.e une conjonction de littéraux, et la recherche sous hypothèses est restreinte à ce cube (dans la suite de ce papier, nous dénotons les cubes en utilisant des crochets, parfois aussi, nous identifions les cubes en considérant les formules qu'ils impliquent). Pendant la recherche sous hypothèses, si l'un des littéraux de celles-ci est mis à faux, alors le problème est prouvé insatisfaisable sous les hypothèses concernées. Dans ce cas, il est possible de traverser récursivement le graphe d'implication afin d'extraire une clause expliquant la raison du conflit. Même si ce problème semble proche du problème SAT classique, un *track* spécial lui a été dédié à la dernière compétition [32] et il existe plusieurs études tentant d'améliorer les solveurs SAT sous hypothèses [4, 26, 29].

3 La gestion de l'arbre

3.1 Transmission

Les performances des approches « diviser pour mieux régner » dépendent surtout de l'efficacité de la division de l'espace de recherche mais aussi de l'assignation des solveurs aux sous-espaces. Même si AMPHAROS est un solveur de type « diviser pour mieux régner », il est important de noter que, contrairement à [33], il n'utilise pas la stratégie de vol de travail (*work stealing*) afin de donner des sous-problèmes aux solveurs. Dans notre cas, la division est faite d'une manière classique comme dans [2, 14]. Plus précisément, notre approche génère des chemins de guidages, réduit aux cubes et couvrant tout l'espace de recherche. De cette manière, le résultat de cette division est un arbre où les nœuds sont des variables et les branches gauches (resp. droites) correspondent à l'assignement de la variable (du nœud de départ de la branche) à vrai (resp. faux). Les solveurs opèrent sur les feuilles (représentées par le symbole NIL) et résolvent (sous hypothèses) la formule initiale restreinte à un cube correspondant au chemin de la racine à la feuille associée. La Fig. 1a montre l'exemple d'un tel arbre contenant trois feuilles (les cubes $[x_1,\neg x_2,x_4]$, $[x_1,\neg x_2,\neg x_4]$ et $[\neg x_1,\neg x_3]$), deux branches fermées (déjà prouvées insatisfaisables) et quatre solveurs (S_1...S_4) travaillant sur ces feuilles.

Comme nous allons le voir dans la section 3.1, dans notre architecture, les solveurs peuvent travailler sur le même cube (comme les solveurs S_1 et S_2 dans la Fig. 1a) et s'arrêter avant de trouver une solution ou une contradiction. Dans AMPHAROS, à chaque fois qu'un solveur partage des informations ou demande un nouveau cube à résoudre, il communique avec un processus dédié appelé le MANAGER. Sa principale mission est de gérer l'arbre de division (les cubes) et la communication entre les solveurs (de type CDCL). Ainsi, quand un solveur prend la décision d'arrêter de résoudre un cube donné (sans l'avoir résolu), il peut demander au MANAGER de l'étendre (voir Sect. 3.3). Un solveur peut aussi arrêter de résoudre un cube donné quand il le prouve insatisfaisable et dans ce cas, un message informe le MANAGER afin de mettre à jour l'arbre en conséquence (voir section 3.4). Dans ces deux cas, à chaque fois qu'un solveur stoppe la résolution d'un cube, il va communiquer avec le MANAGER afin de descendre dans l'arbre et choisir lui-même un nouveau cube à résoudre (potentiellement le même, voir Sect. 3.1). La fin du processus de résolution arrive finalement quand un cube est prouvé satisfaisable ou quand l'arbre complet est prouvé insatisfaisable.

3.2 Initialisation

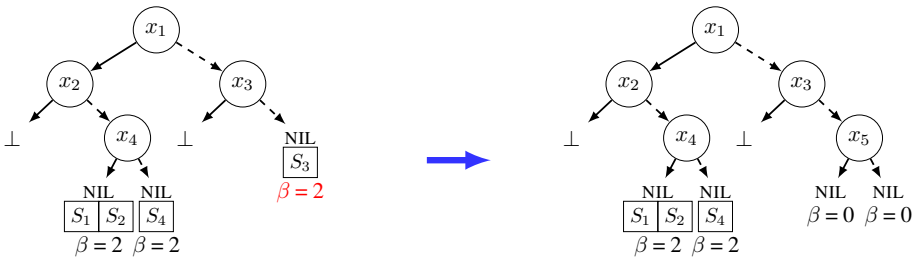
Au départ du processus de recherche, nous initialisons les *workers*. Cette étape permet d'initialiser l'activité des variables (associées à l'heuristique VSIDS [28]) ainsi que leurs polarités et de créer la racine de l'arbre. Pour cela, tous les solveurs essaient de résoudre en concurrence la formule complète pendant un certain nombre de conflits (10,000 dans notre implémentation). Cela correspond à résoudre la formule sans hypothèse (avec un cube vide). Afin de diversifier la recherche, la première descente de chaque solveur est faite aléatoirement (*i.e.* le choix des variables et leurs polarités). Puis, de la même manière que dans [27], le premier solveur atteignant le nombre maximum de conflits communique sa meilleure variable suivant l'heuristique VSIDS au MANAGER. Cette variable devient la racine de l'arbre et tous les solveurs arrêtent leurs recherches concurrentielles afin de demander un cube au MANAGER. Initialement, l'arbre contient donc seulement deux feuilles, *i.e.* les cubes sont restreints à un seul littéral (la variable et son opposé). Dans la figure Fig. 1a, la variable sélectionnée était x_1 et l'ensemble initial des cubes était $\{[x_1],[\neg x_1]\}$.

Comme indiqué au préalable, un solveur peut arrêter la recherche avant d'avoir fini de résoudre un cube. Cette situation se produit quand il n'arrive pas à résoudre le sous-problème associé au cube avant un certain nombre de conflits (fixé à $10{,}000$ ici). Le solveur contacte alors le MANAGER afin de sélectionner un nouveau cube à résoudre (qui peut être le même). L'originalité de notre méthode est de laisser le solveur choisir son cube lui-même parmi tous ceux non résolus dans l'arbre (correspondant aux feuilles NIL). La Fig. 1 montre un diagramme de séquence (Fig. 1b) qui illustre les messages échangés lorsque le solveur S_4 demande un nouveau cube au MANAGER (Fig. 1a).

Un premier message (GO-ROOT) est envoyé par le solveur pour demander la variable associée à la racine de l'arbre. Il reçoit x_1. Par la suite, à chaque étape de sélection d'une branche, le solveur demande les variables des nœuds fils de la variable précédemment reçue (message GIVE-CHILDREN). La réponse est composée de deux triplets : une pour chaque polarité du nœud courant. Chaque triplet est composé, dans cet ordre, de la variable du fils, du nombre de feuilles disponibles (nœuds NIL) et du nombre de solveurs travaillant à cet instant sur ces feuilles. Dans la Fig. 1b, la réponse du MANAGER au premier message GIVE-CHILDREN est le triplet $(x_2,2,2)$ pour la polarité positive de x_1 (la branche gauche contient deux feuilles, deux solveurs (S_1 et S_2)) et le triplet $(x_3,1,1)$ pour la négative.

À ce moment, le solveur va prendre une décision selon les valeurs des triplets : soit il va descendre vers la gauche (affecter positivement la variable courante) soit vers la droite (affecter négativement la variable courante). Par défaut, il choisit la branche possédant un nombre de solveurs inférieur au nombre de feuilles. L'idée est de couvrir le plus de cubes dans l'arbre et ainsi diversifier la position des solveurs. Si cette condition est soit vraie soit fausse pour les deux branches (triplets), alors le solveur sélectionne la branche en fonction de son vecteur de polarité (sa *phase-saving*) [30]. Après avoir sélectionné sa branche, le solveur informe le MANAGER (via le message GO-LEFT ou GO-RIGHT) et affecte le littéral associé comme hypothèse.

Ainsi, dans la Fig. 1, le solveur S_4 affecte x_1 (la racine) positivement (la condition mentionnée précédemment est fausse pour les deux branches). Par la suite, étant donné qu'une des branches liée à x_2 est déjà prouvée insatisfaisable, S_4 n'a pas d'autre alternative que d'affecter x_2 à faux (négativement).



(a) Demande d'extension par le solveur S_3

(b) Extension acceptee par le MANAGER

FIGURE 2 – La figure de gauche représente l'arbre avant l'extension. Comme la valeur de β satisfait le critère d'extension, le MANAGER l'accepte et modifie l'arbre pour obtenir celle de droite.

Puis, il doit mettre x_4 à faux puisque la condition précédente est prise en compte. Enfin, le solveur est arrivé sur une feuille (NIL) et peut commencer à résoudre le cube $[x_1,\neg x_2,\neg,x_4]$.

3.3 L'extension

Initialement, l'arbre contient une seule variable et deux cubes à résoudre (voir Sect. 3.2). Pour diviser la formule originale en un nombre de cubes conséquent, nous proposons d'étendre dynamiquement l'arbre pendant la recherche. Rappelons que nous n'utilisons pas une stratégie de type *work stealing*.

Nous associons à chaque feuille de l'arbre une variable entière β représentant la difficulté supposée du sous-problème associé. À chaque fois qu'un solveur arrête de résoudre un cube (sans trouver de solution), la variable β de la feuille associée à ce cube est incrémentée. De ce fait, plus β est grand, plus le cube associé est potentiellement difficile à résoudre (car beaucoup de solveurs n'ont pas réussi à le résoudre). Notons qu'un même solveur peut incrémenter plusieurs fois la même variable β. À chaque fois qu'un solveur demande un nouveau cube, la valeur β de la feuille associée est examinée. Si elle est plus grande ou égale au nombre de feuilles disponibles (*i.e.* les feuilles NIL) fois un facteur d'extension f_e, alors l'arbre est étendu au niveau de la feuille associée.

L'extension est faite de cette manière : le dernier solveur incrémentant la variable β transmet au MANAGER sa meilleure variable booléenne suivant l'heuristique VSIDS puis deux nouvelles feuilles sont créées afin d'étendre le cube associé. La valeur β des deux nouvelles feuilles est initialisée à 0. Le fait de prendre en compte le nombre de feuilles disponibles permet de gérer le nombre de cubes : plus l'arbre contient de cubes, moins nous aurons d'extensions. De cette manière et contrairement à Cube And Conquer [36], notre approche évite de créer un trop grand nombre de cubes, en prenant compte des cubes déjà prouvés insatisfaisables. Comme une feuille peut contenir plusieurs solveurs, notons qu'après une extension, quelques solveurs peuvent travailler sur des nœuds qui ne sont pas des feuilles.

La Fig. 2 montre l'exemple d'une extension. L'arbre (le même que la Fig. 1) contient 3 feuilles disponibles et quelques solveurs travaillent sur ces feuilles. Quand le solveur S_3 arrête de résoudre le cube $[\neg x_1,\neg x_3]$, la variable associée β (en rouge) est incrémentée et devient égale à 3. La condition permettant l'extension est vérifiée (ici, nous supposons f_e égal à 1) et l'extension est réalisée. Le solveur 3 envoi sa meilleure variable (x_5) et le cube $[\neg x_1,\neg x_3]$ est étendu avec cette variable en générant deux nouveaux cubes. Notons que la valeur β initiatrice de cette extension (en rouge) devient inutile car le nœud associé n'est plus une feuille. Le solveur S_3 est maintenant libre de demander au MANAGER un nouveau cube à résoudre. De plus, dans la prochaine étape, quelque soit le solveur demandant une extension, elle ne sera pas effectuée car le nombre de feuilles disponibles est maintenant égal à 4 (Nous avons supposé dans cette section que f_e est toujours égal à 1, nous discutons de ce facteur dans la section 5.1.1).

3.4 L'élagage

Comme nous utilisons la notion d'hypothèse, quand un cube s'avère être insatisfaisable, le solveur (celui prouvant l'insatisfaisabilité) calcule une clause conflit étant la négation d'un sous-ensemble des littéraux hypothèses. Cette information est transférée au MANAGER afin qu'il calcule un niveau de coupure dans l'arbre. Ainsi, l'arbre est simplifié en conséquence. Remarquons qu'un solveur peut prouver directement l'insatisfaisabilité globale du problème lorsque la clause conflit calculée est vide. De plus, si les deux fils d'un nœud sont insatisfaisables alors ce nœud devient également insatisfaisable. Dans ce cas, ce nœud peut être supprimé et l'insatisfaisabilité est directement associée à la branche du parent. Bien sûr, ce processus est appliqué récursivement jusqu'à l'obtention d'un nœud possédant au moins un fils non-insatisfaisable.

4 L'échange des clauses

4.1 Le partage classique des clauses apprises

Il est bien connu que le partage des clauses apprises améliore sensiblement la performance des solveurs SAT parallèles [22].

Ici, les solveurs partagent également certaines clauses apprises. Cependant, les solveurs ne s'échangent pas directement les clauses entre eux mais ces dernières transitent via le MANAGER.

À chaque fois qu'un solveur atteint un certain nombre de conflits (500 dans notre implémentation), il communique avec le MANAGER pour envoyer et/ou recevoir un ensemble de clauses. Les clauses devant être envoyées sont enregistrées dans une mémoire tampon qui est effacée après chaque communication avec le MANAGER. Les clauses possédant initialement un bon LBD (inférieur ou égal à 2) sont directement mises dans le tampon. D'autres clauses sont également ajoutées, comme dans [6], si elles participent à l'analyse d'un conflit. Cependant, comme nous ne pouvons pas partager autant de clauses que dans SYRUP, seules les clauses qui obtiennent un LBD dynamique inférieur ou égal à 2 avant d'être utilisées deux fois dans l'analyse d'un conflit sont partagées.

Afin de récupérer les clauses importées, les solveurs possèdent chacun trois tampons : `standby`, `purgatory` et `learnt`. Les clauses reçues sont d'abord stockées dans le `standby`. Dans ce tampon, les clauses ne sont pas attachées au solveur [3]. Tous les 4,000 conflits, les clauses sont examinées : elles peuvent être transférées d'un tampon à un autre, être définitivement supprimées ou, enfin, être gardées dans le tampon courant.

Une clause du `standby` peut être transférée dans le `purgatory`. Contrairement au `standby` les clauses du `purgatory` sont attachées au solveur et participent à la propagation. Nous discutons du critère permettant de déplacer une clause du `standby` au `purgatory` dans la Sect. 5.1.2.

De la même manière, une clause du `purgatory` peut être transférée dans les `learnt` si elle est utilisée au moins une fois dans l'analyse d'un conflit. Le tampon temporaire `purgatory` est utilisé pour limiter l'impact des nouvelles clauses dans la stratégie de réduction des clauses apprises.

La stratégie de nettoyage de ces deux tampons supplémentaires (`standby` et `purgatory`) dépend d'un compteur associé à chaque clause. Le compteur est incrémenté à chaque fois que les clauses sont examinées. Si le compteur atteint un certain seuil (14 dans notre implémentation), la clause est supprimée. Notons que le compteur d'une clause est réinitialisé à chaque fois qu'elle change de tampon.

Le MANAGER gère les clauses apprises de tous les solveurs. Les clauses apprises sont stockées dans une queue et le MANAGER vérifie périodiquement si elles sont subsumées ou pas. En pratique, un seul cœur de calcul est dédié au MANAGER. De ce fait, vérifier toutes les clauses en même temps peut s'avérer très coûteux en temps de calcul et peut bloquer les communications entre le MANAGER et les solveurs. Afin d'éviter cette situation, le MANAGER calcule les clauses subsumées par paquet de 1,000 et peut ainsi s'occuper entre temps des communications avec les solveurs. Le MANAGER garde uniquement les clauses apprises qui ne sont pas subsumées dans sa base et les envoient à chaque fois qu'un solveur les demande.

4.2 Les littéraux unitaires sous hypothèses

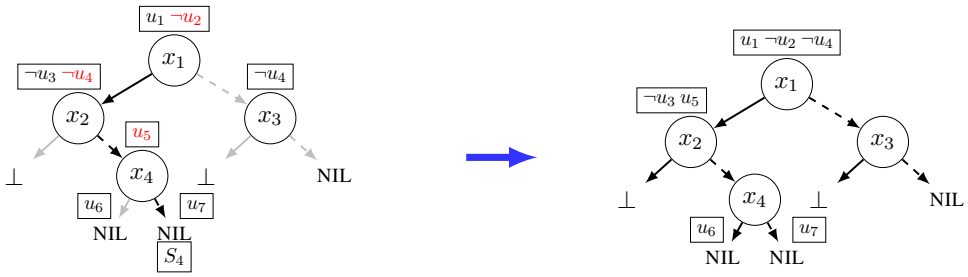
La seconde façon d'échanger des informations dans notre approche est de transférer les littéraux unitaires (qui sont propagés grâce au littéraux provenant des hypothèses) entre les solveurs et le MANAGER. Dans cette section, nous présentons d'où proviennent ces littéraux et comment ils sont échangés et gérés.

Rappelons que chaque solveur travaille sous une hypothèse A (qui peut être vide) représentant le cube à résoudre. Quand un littéral $\ell \notin A$ est propagé grâce à une sous-hypothèse $A' \subseteq A$, cette information peut être transmise au MANAGER afin d'être diffusée aux autres solveurs. Plus précisément, le solveur communique au MANAGER que ℓ peut être propagé avec A'. De l'autre côté, quand un solveur sélectionne une branche (ie un littéral ℓ') durant la transmission d'un cube, il va aussi recevoir un ensemble de littéraux unitaire associé à ℓ' afin de les propager. De ce fait, la transmission d'un cube (voir Sect. 3.1) contient ces messages additionnels.

Par conséquent, le MANAGER s'occupe de décorer l'arbre contenant les chemins de guidages par des ensembles de littéraux unitaires devant être propagés à chaque branche. La figure 3.a montre l'exemple d'un tel arbre. Quand un solveur S_4 demande une branche, il commence par récupérer l'ensemble des littéraux unitaires $\{u_1\}$. Il propage aussi $\neg u_2$ (en rouge) et donne cette information au MANAGER. Il choisit la branche x_1 et récupère le littéral $\neg u_3$ afin de le propager. De la même manière, il propage aussi $\neg u_4$ et apporte ce littéral unitaire sous hypothèses au MANAGER et ainsi de suite.

Comme nous allons le voir dans la partie expérimentation, les littéraux sous hypothèses sont très importants. Ce sont des clauses spéciales qui réduisent clairement l'espace de recherche d'une branche donnée. Par conséquent, le fait qu'un littéral ℓ peut être propagé à partir de A' est pris en compte dans le solveur concerné en ajoutant, dans une base dédiée, une clause composée de la négation de A' et du littéral ℓ' (cette base n'est pas la même que celle des clauses apprises (`learnt`) car elle n'est jamais nettoyée). Remarquons que quand $A' = \emptyset$, le littéral ℓ' est unitaire et est ajouté dans les littéraux unitaires du solveur.

Quand le MANAGER apprend qu'un littéral peut être propagé à partir d'un sous-ensemble de littéraux provenant d'une hypothèse, ce littéral est communiqué pendant la transmission du cube et ajouté dans la dernière branche du nœud associé à cette sous-hypothèse. Dans l'arbre, on peut remonter certains littéraux d'une branche dans d'autres branches plus hautes. Cette situation arrive soit quand une branche est prouvée insatisfaisable, soit quand les deux branches d'un même nœud possèdent le même littéral [9] (comme souligné dans la Fig. 3). Dans le premier cas, tous les littéraux de la branche non-insatisfaisable sont remontés dans la branche père (comme le littéral u_5). Dans le second cas, les littéraux apparaissant dans les deux branches d'un nœud sont remontés dans la branche père (c'est le cas du littéral $\neg u_4$). Ce processus est exécuté récursivement jusqu'à l'obtention d'un point fixe. Remarquons que quand la branche père n'existe pas (cela arrive quand les littéraux sont déplacés à partir d'une branche



(a) S_4 gives $\neg u_2\ \neg u_4\ u_5$ to MANAGER

(b) Unit literals management

FIGURE 3 – La figure de gauche montre un arbre décoré avec les littéraux donnés par S_4 en rouge. Ces littéraux sont remontés, en utilisant deux règles (insatisfaisabilité pour u_5 et littéraux identiques pour $\neg u_4$) afin d'obtenir celle de droite.

de la racine), alors ces littéraux sont prouvés unitaires.

5 Intensification vs Diversification

Quand plusieurs solveurs résolvent en concurrence un problème, ils peuvent faire des recherches redondantes. Identifier une telle situation permettrait de pouvoir modifier la stratégie des solveurs afin de diversifier leurs recherches. Toutefois, à cause du partage des clauses apprises entre les solveurs, explorer différents espaces de recherche pourrait être un handicap. Dans une telle situation, focaliser tous les solveurs sur le même espace de recherche pourrait être essentiel (intensification).

Ce paradigme, appelé dilemme d'intensification/diversification, a déjà été étudié dans le contexte des solveurs SAT de type *portfolio*. Il peut être traité soit statiquement, en utilisant plusieurs solveurs avec des stratégies opposées [1, 22, 31], soit dynamiquement, en modifiant la stratégie des solveurs pendant la recherche. Cependant, savoir quand la recherche d'un solveur doit être intensifiée ou diversifiée n'est pas facile et seulement quelques publications essaient de traiter ce problème [19, 20]. Dans [19], une architecture maître/esclave est proposée. Les maîtres tentent de résoudre le problème original (assurant la diversification), tandis que les esclaves intensifient la stratégie de leur maître. Dans [20], une mesure pour estimer le degré de la redondance entre deux solveurs est présentée. Elle considère que deux solveurs sont proches quand ils ont approximativement le même vecteur de polarité (*phase-saving*). Le processus de diversification consiste alors à modifier la manière de choisir la polarité des variables de décision.

À notre connaissance, aucun critère permet d'identifier si plusieurs solveurs exécutent une recherche redondante mis à part la mesure sur la polarité mentionné précédemment [20]. Malheureusement, ce critère n'est pas applicable avec beaucoup de solveurs (initialement proposée pour un *portfolio* de quatre solveurs). C'est pourquoi un critère plus scalable est requis.

5.1 Évaluation du degré de redondance

Nous proposons de mesurer le degré de redondance en prenant en compte le nombre de clauses redondantes partagées entre les solveurs. Nous utilisons une liste afin de mémoriser depuis le début le nombre de clauses reçues (st_r) et une autre afin de mémoriser le nombre de clauses gardées (st_k). Les clauses gardées sont celles qui n'ont pas été supprimées durant la vérification des clauses subsumées. Quand un solveur revient vers le MANAGER pour partager des clauses (tous les 1,000), le nombre de clauses reçues (resp. gardées) depuis le début est sauvegardé dans st_r (resp. st_k) par le MANAGER.

La *redundancy shared clauses measure*, en bref $rscm$, est définie pour une étape t suivant un intervalle glissant de taille m (20,000 dans nos expérimentations) comme le ratio entre le nombre de clauses reçues durant les $t-m$ mises à jour de st_r et le nombre de clauses gardées durant ce même temps. Plus précisément, nous avons $\forall j<0$, $st_r[j]=st_k[j]=0$:

$$rscm_t = \frac{st_r[t]-st_r[t-m]}{st_k[t]-st_k[t-m]}, \text{ si } st_k[t]-st_k[t-m]\neq 0$$
$$rscm_t = st_r[t]-st_r[t-m], \text{ sinon} \quad (1)$$

Tout d'abord, notons que lorsque plusieurs solveurs travaillent sur le même espace de recherche, il y a une forte probabilité que les clauses apprises par les différents solveurs soient redondantes. Cela signifie que le nombre de clauses subsumées est important, et donc, d'avoir un $rscm$ élevé. Inversement, lorsque les solveurs sont diversifiés dans l'espace de recherche, il y a une forte probabilité d'avoir des clauses non redondantes, et donc, d'avoir un $rscm$ faible. Par conséquent, un $rscm$ faible indique que le solveur doit intensifier la recherche, tandis qu'une valeur $rscm$ élevée signifie que les solveurs doivent diversifier leur recherche.

Il y a plusieurs façons de diversifier ou d'intensifier les solveurs (clauses apprises, les heuristiques des solveurs, ...). Dans AMPHAROS, nous choisissons de résoudre le dilemme d'intensification/diversification en contrôlant deux critères : la

manière dont l'arbre est étendu (voir Sect. 3.3) et le nombre de clauses passant du `standby` aux `purgatory` (voir Sect. 4.1). Ainsi, pour nous, diversifier (resp. intensifier) la recherche consiste à augmenter (resp. diminuer) ces deux paramètres.

Peu de clauses subsumées ($rscm$ est petit)	**Beaucoup de clauses subsumées** ($rscm$ est grand)
Réduit l'extension	Favorise l'extension
Augmente les clauses importées	Limite les clauses importées
Intensification	**Diversification**

5.1.1 L'extension guidée par le $rscm$

Tout d'abord, remarquons que chaque chemin de la racine à une feuille représente un ensemble unique de littéraux qui divise l'espace de recherche d'une manière déterministe. Ainsi, plus l'arbre est grand, plus la probabilité d'exécuter deux solveurs dans deux sous-problèmes distincts augmente. Afin de contrôler cette croissance, nous définissons le facteur d'extension :

$$fe_t = \frac{1000}{{rscm_t}^3} \quad (2)$$

Rappelons que ce facteur d'extension est utilisé afin d'accroître ou de diminuer l'extension de l'arbre et est associé au nombre de cubes disponibles. Par conséquent, plus la valeur $rscm_t$ est petite (resp. grande), plus la valeur fe sera grande (resp. petite), et donc plus l'extension de l'arbre sera lente (resp. rapide). Notons que le ${rscm_t}^3$ (au cube) permet de diminuer fe rapidement, tandis que la division (par 1000) permet de le limiter au cas où les solveurs seraient en concurrence. Afin d'empêcher l'arbre de s'étendre trop, nous avons également limité la valeur de fe a un maximum de 10.

5.1.2 Du `standby` au `purgatory`

Quand une clause est reçue par un solveur, elle peut être subsumée par celles déjà présentes. Plus il y a de clauses subsumées, plus il est probable d'en avoir. Ainsi, il semble naturel que le nombre de clauses acceptées par un solveur (celles passant du `standby` au `purgatory`) augmente (resp. diminue) quand la valeur $rscm$ diminue (augmente).

Dans AMPHAROS, les clauses fraîchement reçues ne sont pas directement attachées au solveur. Ainsi, nous devons choisir un critère de sélection indépendant afin d'activer ces clauses. Pour contrôler la quantité de clauses passant du `standby` au `purgatory`, nous utilisons la notion de psm ([3]) et déjà utilisée dans le solveur de type *portfolio* PENELOPE [1]. Rappelons que le psm d'une clause représente le nombre de littéraux qui sont affectés à vrai par le vecteur de polarité (*phase-saving*).

Ensuite, afin d'augmenter/diminuer le nombre de clauses attachées (et alors transférées) dans le `purgatory`, un critère fusionnant les valeurs psm et $rscm$ est proposé. Ce critère est justifié par l'observation de [3]. Les auteurs montrent expérimentalement que les clauses possédant un petit psm ont plus de chance d'entrer en conflit ou d'être utilisées pendant la recherche. De ce fait, dans notre méthode, une clause est autorisée à passer du `standby` au `purgatory` quand sa valeur psm est inférieure ou égale à $\lfloor \frac{psm_{max}}{rscm_t} \rfloor$, où psm_{max} correspond au psm maximum accepté (fixé à 6 dans nos expérimentations). En conséquence, les clauses avec un psm de zéro sont systématiquement acceptées quelles que soient la valeur $rscm$. Tandis que les clauses possédant une valeur psm élevée seront acceptées si et seulement s'il y a peu de clauses subsumées au niveau du MANAGER.

6 Évaluation expérimentale

Nous évaluons AMPHAROS sur les 100 instances du *parallel track* de la SAT-RACE 2015 [32]. Durant cette compétition, 53 (resp. 33) instances ont été prouvées satisfaisables (resp. insatisfaisables) par au moins un des solveurs et 14 instances sont restées non résolues. Les expérimentations conduites dans cet article sont réalisées sur des bi-processeurs Intel XEON X5550 4 cœurs à 2.66 GHz avec 8Mo de cache et 32 Go de RAM, sous Linux CentOS 6 (pour un total de 64 cœurs). La limite de temps alloué pour résoudre une instance est de 1200 secondes (temps réel). Pour les expérimentations faites sur 64 cœurs, nous utilisons donc deux machines. Tous les fichiers de *log*, des diagrammes de dispersion et des cactus additionnels sont disponibles à l'adresse `http://www.cril.univ-artois.fr/ampharos`.

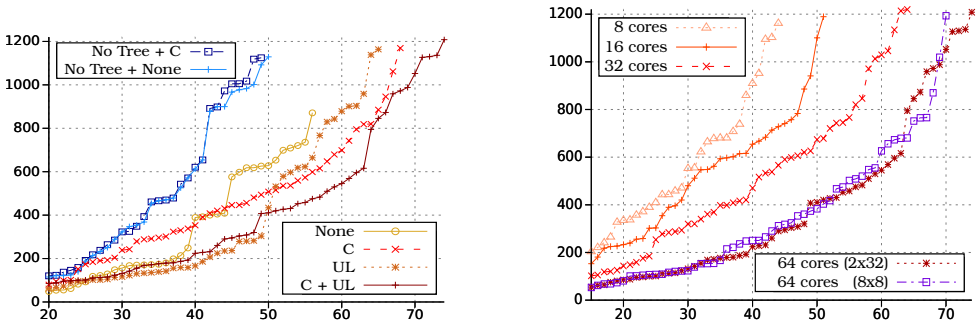
6.1 Gestion des communications

Comme dans AMPHAROS un grand nombre de messages sont échangés entre le MANAGER et les solveurs, la gestion des communications doit être efficace. Ainsi, nous avons choisi la librairie *open source* Open MPI (pour *Message Passing Interface implementation*) afin de gérer les communications.

Le goulot d'étranglement imposé par le fait que le MANAGER doit à la fois communiquer avec tous les solveurs et calculer les clauses subsumées a été un problème majeur. Afin d'éviter que les solveurs attendent trop longtemps sans travailler, un tourniquet associé à une écoute non-bloquante des solveurs par le MANAGER a été mis en place. De plus, comme nous identifions les clauses subsumées reçues et que ceci peut être coûteux, nous ne les traitons pas dés leur réception mais petit à petit par paquet de 1,000 clauses.

6.2 Configuration

AMPHAROS est un outil modulaire qui permet d'ajouter facilement de nouveaux types de solveurs. Pour ces expérimentations, trois solveurs SAT séquentielles ont été utilisés : GLUCOSE [5], MINISAT [15] et MINISATPSM [3]. Seuls quelques changements ont été mis en œuvre dans ces solveurs. Afin de gérer les interactions avec le MANAGER, tous les solveurs doivent implémenter une interface en C++. Cette interface permet de regrouper les méthodes engendrant des communications ou celles faisant intervenir des ajouts dans les solveurs (les mémoires tampons `standby` et `purgatory`). Le noyau des



(a) Evaluation des composants de Ampharos.

(b) Evaluation de la scalabilité

FIGURE 4 – Comparaison de plusieurs versions de AMPHAROS et évaluation de la scalabilité

solveurs a également été modifié afin d'éviter de réinitialiser certaines heuristiques à chaque fois qu'un cube doit être résolu (*restart*, nettoyage de la base des clauses apprises, ...). Par ailleurs, comme pour la version de GLUCOSE présentée dans [4], quand un solveur redémarre, il n'a pas besoin de faire un saut au niveau de décision 0, mais au niveau de la dernière hypothèse. Les clauses transférées du `purgatory` au `learnt` sont simplement incorporées dans la base des clauses apprises du solveur comme si elles étaient apprises par lui-même.

6.3 Résultats

L'évaluation expérimentale est divisée en trois parties. Premièrement, nous évaluons les différentes approches d'AMPHAROS. Puis, nous étudions la scalabilité de notre solveur. Finalement, nous comparons AMPHAROS avec quelques solveurs de l'état de l'art.

6.3.1 L'impact de chaque composant

Les bénéfices des trois composants optionnels (décomposition via un arbre, (**Tree**), partage des clauses (**C**), et échange des littéraux unitaires (**UL**)) ont été étudiés expérimentalement. À cette fin, plusieurs versions d'AMPHAROS ont été exécutées sur 64 cœurs. Ces expériences, présentées dans la Fig. 4a, montre une amélioration progressive lorsque chacune de ces options a été prises en compte cumulativement. Plusieurs observations peuvent être déduites du cactus.

Premièrement, que quelle que soit la combinaison des options utilisées, AMPHAROS est plus efficace avec la décomposition via un arbre. Les versions concurrentielles (pouvant être considérées comme *portfolio*) (**No Tree**) avec (**C**) ou sans (**none**) le partage des clauses, résolvent systématiquement moins d'instances que les versions de type « diviser pour mieux régner ». Cela montre l'importance de la manière de résoudre le dilemme intensification/diversification en utilisant une décomposition via un arbre dans AMPHAROS (**Tree Yes**).

Deuxièmement, Les résultats montrent l'importance du partage des informations entre les solveurs quand les cubes sont activés. La version d'AMPHAROS qui n'échange aucune information résout systématiquement moins d'instances que celles qui en échangent ((**C**) et/ou (**UL**)). Quand nous comparons séparément ces deux types d'échange, nous observons que le partage des clauses apprises permet d'améliorer les résultats sur les problèmes insatisfaisables (comme attendu). Toutefois, activer cette option rend le solveur plus lent sur les problèmes faciles (résolus en moins de 600 secondes). Ceci peut être expliqué par le fait que les communications engendrées par les clauses partagées peuvent diminuer la vitesse de notre solveur sur ces instances « faciles ».

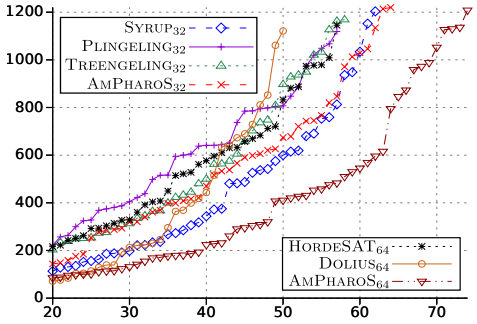
Finalement, nous mettons en évidence une synergie très positive entre les types d'échange. Même si le partage des clauses peut dramatiquement réduire les performances du solveur sur les instances faciles, la combinaison de ce composant avec l'échange des littéraux unitaires sous hypothèses (**C + UL**) permet de donner l'amélioration la plus significative à la fois en nombre d'instances résolues et sur le cactus 4a. À partir de maintenant, quand nous parlerons d'AMPHAROS, il s'agira de la version **C + UL**.

6.3.2 Évaluation de la scalabilité

Afin d'évaluer la scalabilité d'AMPHAROS, nous le lançons sur 8, 16, 32 et 64 cœurs. La Fig. 4b nous donne le nombre d'instances résolues par rapport au temps sur ces différentes configurations. Ce cactus montre clairement que notre approche est très scalable. La version sur 64 cœurs résout 49 SAT et 25 UNSAT, ce qui est 15%, 45% et 70% fois plus d'instances que celles sur, respectivement, 32 (44 SAT et 20 UNSAT), 16 (36 SAT et 15 UNSAT) et 8 (33 SAT et 11 UNSAT) cœurs. De plus, nous montrons aussi l'efficacité de notre solveur dans un environnement plus distribué avec 8 ordinateurs de 8 cœurs pour un total de 64 cœurs (courbe 8×8 de la Fig. 4b). Cette version résout 46 SAT et 24 UNSAT. Comme nous limitons le nombre de messages, nous obtenons des résultats similaires. La faible différence entre les courbes $8x8$ et 64 s'explique par l'indéterminisme de notre approche et reste acceptable.

Solveur	#thr.	SAT	UNS	Total
AMPHAROS	32	**44**	20	**64**
SYRUP	32	36	**26**	62
TREENGELING	32	38	20	58
PLINGELING	32	31	**26**	57
AMPHAROS	64	**49**	**25**	**74**
HORDESAT	64	33	24	57
DOLIUS	64	33	17	50

(a) Tableau global



(b) Nombre d'instances par rapport au temps (secondes)

FIGURE 5 – Comparaisons d'AMPHAROS vs l'état de l'art des solveurs SAT parallèles. Le Tab. 5a donne les résultats de chaque solveur suivant le nombre de *threads* (**#thr.**).

6.3.3 AMPHAROS vs l'état de l'art

Nous avons choisi d'évaluer notre approche avec les trois meilleurs solveurs du *parallel track* de la SAT-RACE 2015 [32] sur 32 cœurs puis avec deux solveurs compatibles avec une configuration distribuée sur 64 cœurs afin de respecter la majorité des travaux existants. Les trois solveurs de la compétition (suivant leur rang) sont : SYRUP [6], TREENGELING et PLINGELING [10]. Nous ne pouvons les lancer que sur 32 cœurs car ces solveurs ne sont pas distribués et nous n'avons pas de processeur contenant 64 cœurs mais que deux machines de 32 cœurs. De ce fait, nous comparons aussi notre solveur avec d'autres solveurs SAT distribués sur 64 cœurs : le solveur parallèle de type *work stealing* DOLIUS [2] et le solveur de type *portfolio* HORDESAT [7].

Considérons tout d'abord, les expérimentations sur 32 cœurs. AMPHAROS résout plus d'instances que les autres solveurs (tableau 5a). Il est le meilleur solveur sur les instances satisfaisables et résout autant d'instances insastisfaisables que TREENGELING (qui lui aussi est un solveur de type « diviser pour mieux régner »). Par rapport à SYRUP et PLINGELING, notre solveur est meilleur sur les instances satisfaisables mais moins efficace sur celles insatisfaisables. Ceci peut partiellement s'expliquer par le fait qu'AMPHAROS résout essentiellement les problèmes insatisfaisables en réfutant complètement l'arbre (*i.e.* en fermant toutes les branches). En conséquence, il semble que sur 32 cœurs, nous n'avons pas assez de *workers* pour atteindre ce but dans le temps imparti.

En considérant les temps réels du cactus 5b, nous pouvons observer qu'AMPHAROS est plus rapide que TREENGELING et PLINGELING mais plus lent que SYRUP. Nous pouvons expliquer ceci par le fait que SYRUP résout plusieurs instances d'une même famille très rapidement (la famille des instances `6s`).

Dans un second temps, nous pouvons observer sur 64 cœurs que notre approche est très compétitive. AMPHAROS est significativement meilleur que DOLIUS et HORDESAT. De plus, il est important de noter que, durant la compétition, SYRUP (le gagnant du *parallel track*) a utilisé seulement 32 cœurs sur les 64 disponibles. En conséquence, il est possible d'admettre, par transitivité, qu'AMPHAROS serait plus efficace que TREENGELING et PLINGELING sur 64 cœurs. Plus important, nous pouvons voir sur le cactus 5b qu'AMPHAROS est vraiment efficace car il résout plus d'instances plus rapidement.

Notons que ces solveurs sont indéterministes. Pour être juste, nous avons fait tourner une seule fois chaque solveur et reporté les résultats obtenus (comme cela est fait à la SAT RACE 2015). Pour conclure cette section, notons qu'une évaluation expérimentale de la valeur $rscm$ est disponible sur le site internet d'AMPHAROS (voir la section 3.3 pour le lien).

7 Conclusion

Notre objectif principal est de déployer un solveur SAT à travers le *cloud*. Ainsi, ce papier est la première brique d'un édifice beaucoup plus vaste dans lequel nous avons de nombreuses perspectives. Nous comptons exécuter plusieurs MANAGERS et les solveurs pourrons migrer d'un MANAGER à un autre. Prendre des heuristiques distinctes afin de sélectionner les variables des arbres et ainsi créer dans la recherche plusieurs divisions orthogonales apportera sans doute une diversification beaucoup plus efficace. Plusieurs possibilités récentes comme la notion des littéraux bloqués pour la résolution du problème SAT peuvent être utilisées à cette fin [25, 12]. Finalement, nous pouvons aussi améliorer les performances de notre approche sur les problèmes insatisfaisables en faisant plus attention aux clauses partagées.

Références

[1] G. Audemard, B. Hoessen, S. Jabbour, JM. Lagniez, and C. Piette. Revisiting clause exchange in parallel SAT solving. In *SAT*, pages 200–213, 2012.

[2] G. Audemard, B. Hoessen, S. Jabbour, and C. Piette. An effective distributed d&c approach for the satisfiability problem. In *PDP*, pages 183–187, 2014.

[3] G. Audemard, JM. Lagniez, B. Mazure, and L. Sais. On freezing and reactivating learnt clauses. In *SAT*, pages 188–200, 2011.

[4] G. Audemard, JM. Lagniez, and L. Simon. Improving glucose for incremental SAT solving with assumptions : Application to MUS extraction. In *SAT*, pages 309–317, 2013.

[5] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI*, pages 399–404, 2009.

[6] G. Audemard and L. Simon. Lazy clause exchange policy for parallel SAT solvers. In *SAT*, pages 197–205, 2014.

[7] T. Balyo, P. Sanders, and C. Sinz. Hordesat : A massively parallel portfolio SAT solver. In *SAT*, pages 156–172, 2015.

[8] A. Belov, I. Lynce, and J. Marques-Silva. Towards efficient MUS extraction. *AI Commun.*, 25(2) :97–116, 2012.

[9] D. Le Berre. Exploiting the real power of unit propagation lookahead. *Electronic Notes in Discrete Mathematics*, 9 :59–80, 2001.

[10] A. Biere. Lingeling, Plingeling and Treengeling entering the SAT competition 2013. In *Proc. of SAT Competition*, page 51, 2013.

[11] A. Biere. Yet another local search solver and lingeling and friends entering the sat competition 2014. In *Proc. of SAT competition*, 2014.

[12] J.p Chen. Minisat bcd and abcdsat : Solvers based on blocked clause decomposition. In *SAT RACE 2015 solvers description*, 2015.

[13] W. Chrabakh and R. Wolski. The gridsat portal : a grid web-based portal for solving satisfiability problems using the national cyberinfrastructure. *Concurrency and Computation : Practice and Experience*, 19(6) :795–808, 2007.

[14] G. Chu, P. Stuckey, and A. Harwood. Pminisat : a parallelization of minisat 2.0. Technical report, SAT Race, 2008.

[15] N. Eén and N. Sörensson. An extensible sat-solver. In *SAT*, pages 502–518, 2003.

[16] Y. Feldman, N. Dershowitz, and Z. Hanna. Parallel multithreaded satisfiability solver : Design and implementation. *Electr. Notes Theor. Comput. Sci.*, 128(3) :75–90, 2005.

[17] E. Goldberg and Y. Novikov. Berkmin : A fast and robust sat-solver. *Discrete Applied Mathematics*, 155(12) :1549–1561, 2007.

[18] C. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.

[19] L. Guo, Y. Hamadi, S. Jabbour, and L. Sais. Diversification and intensification in parallel SAT solving. In *CP*, pages 252–265, 2010.

[20] L. Guo and JM. Lagniez. Dynamic polarity adjustment in a parallel SAT solver. In *ICTAI*, pages 67–73, 2011.

[21] Y. Hamadi, S. Jabbour, and L. Sais. Control-based clause sharing in parallel SAT solving. In *IJCAI*, pages 499–504, 2009.

[22] Y. Hamadi, S. Jabbour, and L. Sais. Manysat : a parallel SAT solver. *JSAT*, 6(4) :245–262, 2009.

[23] M. Heule, O. Kullmann, S. Wieringa, and A. Biere. Cube and conquer : Guiding CDCL SAT solvers by lookaheads. In *Haifa Verification Conference*, pages 50–65, 2011.

[24] A. Hyvärinen, T. Junttila, and I. Niemelä. Partitioning SAT instances for distributed solving. In *LPAR*, pages 372–386, 2010.

[25] M. Järvisalo, A. Biere, and M. Heule. Blocked clause elimination. In *TACAS*, pages 129–144, 2010.

[26] JM. Lagniez and A. Biere. Factoring out assumptions to speed up MUS extraction. In *SAT*, pages 276–292, 2013.

[27] R. Martins, V. M. Manquinho, and I. Lynce. Improving search space splitting for parallel SAT solving. In *ICTAI*, pages 336–343, 2010.

[28] M. W. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *DAC*, pages 530–535. ACM, 2001.

[29] A. Nadel and V. Ryvchin. Efficient SAT solving under assumptions. In *SAT*, pages 242–255, 2012.

[30] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In *SAT*, pages 294–299, 2007.

[31] O. Roussel. ppfolio. `http://www.cril.univ-artois.fr/~roussel/ppfolio`.

[32] SAT-race, 2015. `http://baldur.iti.kit.edu/sat-race-2015/`.

[33] T. Schubert, M. Lewis, and B. Becker. Pamiraxt : Parallel SAT solving with threads and message passing. *JSAT*, 6(4) :203–222, 2009.

[34] A. Semenov and O. Zaikin. Using monte carlo method for searching partitionings of hard variants of boolean satisfiability problem. In *PaCT*, pages 222–230, 2015.

[35] J. P. Marques Silva and K. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.

[36] P. van der Tak, M. Heule, and A. Biere. Concurrent cube-and-conquer. *CoRR*, abs/1402.4465, 2014.

[37] H. Zhang, M. Bonacina, and J. Hsiang. PSATO : a distributed propositional prover and its application to quasigroup problems. *J. Symb. Comput.*, 21(4) :543–560, 1996.

Exploiter les définitions pour le comptage de modèles

Jean-Marie Lagniez[1] **Emmanuel Lonca**[1] **Pierre Marquis**[1]

[1] Université d'Artois, CRIL-CNRS UMR 8188
{lagniez,lonca,marquis}@cril.fr

Résumé

Nous présentons une nouvelle technique pour le comptage de modèles propositionnels qui exploite les définitions de variables, *i.e.* la détection de portes logiques impliquées par la formule Σ fournie en entrée. De telles portes peuvent être utilisées pour simplifier la formule Σ lors d'une phase de prétraitement sans modifier son nombre de modèles, mais en rendant le comptage de ceux-ci plus aisé. Contrairement à d'autres approches de l'état de l'art, nous exploitons simplement l'existence de telles portes, sans besoin de les expliciter. Notre prétraitement se décompose en deux phases : dans un premier temps, nous calculons une bipartition $\langle I, O\rangle$ des variables de Σ telles que les variables de O soient définies dans Σ à partir des variables de I ; dans un second temps, nous éliminons des variables de O dans Σ. Nous montrons l'importance des bénéfices empiriques pouvant être apportés par notre prétraitement pour le comptage de modèles.

Abstract

We present a new preprocessing technique for propositional model counting. This technique leverages definability, i.e., the ability to determine that some gates are implied by the input formula Σ. Such gates can be exploited to simplify Σ without modifying its number of models. Unlike previous approaches, we only take advantage of the existence of gates, but we do not need to make the gates explicit. Our preprocessing technique thus consists of two phases : computing a bipartition $\langle I, O\rangle$ of the variables of Σ where the variables from O are defined in Σ in terms of I, then eliminating some variables of O in Σ. Our large scale experiments clearly show the computational benefits which can be achieved by taking advantage of our preprocessing technique for model counting.

1 Introduction

Le comptage de modèles propositionnels (aussi connu sous le nom de problème #SAT) est le problème qui consiste à compter combien une formule d'entrée Σ admet de modèles. Ce problème (et sa généralisation pondérée) est central en intelligence artificielle (IA), de par son utilisation pour l'inférence probabiliste [31, 8, 1, 9] ou la planification [27, 12], mais aussi en dehors du champ de l'IA ; il est par exemple utilisé pour la recherche de vulnérabilité de circuits électroniques [14].

#SAT est un problème d'une complexité théorique élevée (#P-complet), plus difficile en théorie et en pratique que le problème SAT. Son importance explique l'effort fourni par la communauté durant ces dernières décennies pour développer de nouveaux algorithmes et des logiciels pour compter de manière exacte ou approchée le nombre de modèles de formules de plus en plus grandes [29, 4, 15].

Dans cet article, nous présentons un nouveau prétraitement, basé sur l'identification de définitions, pour améliorer l'efficacité des compteurs exacts de modèles. Les méthodes de prétraitement sont aujourd'hui considérées comme des moyens efficaces pour améliorer divers prouveurs SAT ou QBF [5, 32, 13, 28, 16, 17, 19, 18], et on les trouve donc implantées dans les prouveurs de l'état de l'art (on peut citer, parmi d'autres, le préprocesseur `SatELite` [13] dans `Glucose` [2], le préprocesseur interne de `Lingeling` [7], ou encore le préprocesseur `Coprocessor` [24] utilisé dans `Riss` [25]).

Notre approche se base sur celle de [21], qui décrit divers prétraitements pour améliorer les performances du calcul de nombre de modèles d'une formule Σ, et présente en particulier des prétraitements opérant par *détection et remplacement de portes logiques*. L'idée générale de cette approche est que toute variable y de

Σ définie dans Σ en fonction d'un ensemble d'autres variables $X = \{x_1, \ldots, x_k\}$ peut être remplacée par sa définition Φ_X sans modifier le nombre de modèles de Σ. Plus précisément, pour toute affectation des variables de X, soit cette affectation est inconsistante avec Σ, soit tout modèle étendant cette affectation partielle donne la même valeur de vérité à y. Dans [21], les portes logiques considérées (équivalence de littéraux, portes ET, OU, XOR) permettent au préprocesseur `pmc` proposé par Lagniez et Marquis de conduire à des gains très importants en terme de temps de calcul lorsque les portes détectées par la combinaison de prétraitements nommée $\#eq$ sont remplacées par leurs définitions. Cependant, `pmc` reste limité au sens où il ne permet de ne traiter qu'un nombre réduit de portes logiques.

Pour aller au delà de cette limitation, notre nouvelle technique de prétraitement tend à s'attaquer au problème de manière beaucoup plus agressive, avec l'idée *qu'il n'est pas nécessaire d'identifier les portes logiques mais qu'il peut être suffisant de déterminer que ces portes existent*. Plus précisément, il se révèle suffisant de déterminer qu'il existe des relations de définitions entre variables ; il n'est pas nécessaire de déterminer comment sont définies ces relations. Cette distinction est de première importance dans la mesure où, bien que l'espace de recherche des définitions potentielles Φ_X soit très important (2^{2^k} éléments à l'équivalence logique près, quand X contient k variables), la taille d'une définition explicite Φ_X de y dans Σ n'est pas, dans le cas général, bornée par un polynôme en $|\Sigma|+|X|$ sous l'hypothèse $\mathsf{NP} \cap \mathsf{coNP} \not\subseteq \mathsf{P/poly}$, considérée usuellement adoptée en théorie de la complexité [23].

Nous décrivons ainsi dans la suite un nouveau préprocesseur, `B + E`, qui associe à toute formule `CNF` d'entrée Σ une formule `CNF` Φ qui admet le même nombre de modèles mais qui est au moins aussi « simple » au regard du nombre du nombre de variables et de la taille de la formule [1].

L'algorithme `B + E` consiste en deux étapes : `B` qui calcule une *Bipartition* $\langle I, O\rangle$ des variables de Σ telle que toute variable de O soit définie dans Σ par des variables de I, et `E` dont le but est d'*Éliminer* des variables de O dans Σ. Notre contribution inclut la présentation des algorithmes `B` et `E`, une preuve de la correction de notre préprocesseur, et des résultats expérimentaux montrant les gains fournis par l'utilisation de `B + E` comme préprocesseur à un compteur exact de modèles, en lieu et place de `pmc`. Les instances de tests, le code de `B + E`, et les résultats détaillés de notre étude empirique sont disponibles en ligne à partir de `www.cril.fr/KC/`.

Dans la suite de cet article, nous rappelons d'abord la notion de définition en logique propositionnelle et quelques résultats clés sur le sujet. Nous présentons ensuite le préprocesseur `B + E` et la preuve de sa correction. Nous décrivons l'étude expérimentale réalisée, qui montre les gains apportés par `B + E` (en particulier, par rapport à `pmc`), avant de conclure et de lister quelques perspectives de travaux futurs.

2 De la définissabilité propositionnelle

Commençons par quelques préliminaires sur la logique propositionnelle. Soit $PROP_{PS}$ le langage propositionnel (classique) défini de manière inductive sur l'ensemble fini de variables propositionnelles PS, l'ensemble des connecteurs logiques usuels ($\neg$, $\vee$, $\wedge$, $\leftrightarrow$, ...) et les constantes booléennes $\top$ et $\bot$; ces formules sont interprétées de manière classique. Pour toute formule Σ de $PROP_{PS}$, l'ensemble des variables de PS apparaissant dans Σ est noté $Var(\Sigma)$, le nombre de modèles de Σ sur $Var(\Sigma)$ est noté $\|\Sigma\|$. Un littéral ℓ est soit une variable ($\ell = x$) de PS soit la négation d'une variable ($\ell = \neg x$). Quand ℓ est un littéral, $var(\ell)$ est la variable sur laquelle est construit ce littéral. Un terme est une conjonction de littéraux ou $\top$, une clause est une disjonction de littéraux ou $\bot$. Une formule `CNF` est une conjonction de clauses. Étant donné un sous-ensemble de variables $X \subseteq PS$, un terme canonique γ_X sur X est un terme cohérent dans lequel toutes les variables de X apparaissent (sous la forme d'un littéral positif ou négatif, *i.e.* une variable niée). On note $\exists X.\Sigma$ l'oubli des variables de X dans Σ, *i.e.* toute formule équivalente à la conséquence logique la plus forte de Σ qui est indépendante des variables de X [22].

On rappelle maintenant les deux formes (équivalentes) sous lesquelles le concept de définissabilité propositionnelle peut être rencontré dans la littérature.

Définition 1 (définition implicite) *Soit $\Sigma \in PROP_{PS}$, $X \subseteq PS$ et $y \in PS$.* Σ définit de manière implicite *y à partir de X si et seulement si pour tout terme canonique γ_X sur X, on a $\gamma_X \wedge \Sigma \models y$ ou $\gamma_X \wedge \Sigma \models \neg y$.*

Définition 2 (définition explicite) *Soit $\Sigma \in PROP_{PS}$, $X \subseteq PS$ et $y \in PS$.* Σ définit explicitement *y à partir de X si et seulement si il existe une formule $\Phi_X \in PROP_X$ telle que $\Sigma \models \Phi_X \leftrightarrow y$. Dans ce cas, Φ_X est appelée* définition (ou porte) de y sur

1. La restriction imposant à la formule d'entrée d'être sous forme normale conjonctive est faible dans la mesure où la transformation de Tseitin [34] permet de transformer n'importe quel circuit propositionnel en une formule `CNF` admettant le même nombre de modèles (cet encodage procédant justement par ajout de portes logiques). Notons par ailleurs que les formules `CNF` sont les formules d'entrée des compteurs de modèles de l'état de l'art, comme `Cachet` [30], `C2D` [10, 11] ou `sharpSAT` [33].

X dans Σ, *y est la variable de sortie de la porte, et X est l'ensemble des variables d'entrée.*

Exemple 1 *Soit Σ la formule* CNF *constituée de la conjonction des clauses suivantes :*

$$\begin{array}{lll} a \vee b, & \neg a \vee \neg b \vee d, & a \vee e, \\ a \vee c \vee \neg e, & \neg a \vee \neg c \vee d, & b \vee c \vee e, \\ a \vee \neg d, & \neg a \vee \neg b \vee c \vee \neg e, & \neg b \vee \neg c \vee e, \\ b \vee c \vee \neg d, & \neg a \vee b \vee \neg c \vee \neg e. & \end{array}$$

Les variables d et e sont définies implicitement dans Σ à partir de $X = \{a, b, c\}$. On peut par exemple vérifier que pour le terme canonique $\gamma_X = a \wedge b \wedge \neg c$, on $\gamma_X \wedge \Sigma \models d \wedge \neg e$; pour $\gamma'_X = \neg a \wedge \neg b \wedge \neg c$, $\gamma'_X \wedge \Sigma$ est incohérent. On peut aussi vérifier que d et e sont définies explicitement dans Σ à partir de $X = \{a, b, c\}$; plus précisément Σ implique les équivalences suivantes :

$$d \leftrightarrow (a \wedge (b \vee c)) \text{ et } e \leftrightarrow (\neg a \vee (b \leftrightarrow c)).$$

Le fait que les mêmes variables soient définies à la fois implicitement et explicitement n'est pas fortuit, comme l'explique le théorème suivant, dû à Beth [6], et restreint ici à la logique propositionnelle.

Théorème 1 *Soit $\Sigma \in PROP_{PS}$, $X \subseteq PS$ et $y \in PS$. Σ définit implicitement y à partir de X si et seulement si Σ définit explicitement y à partir de X.*

Puisque ces deux notions de définissabilité coïncident, nous dirons souvent dans la suite que y est défini à partir de X dans Σ, sans préciser s'il s'agit de définissabilité implicite ou explicite.

Une conséquence intéressante du théorème de Beth, dans notre optique de prétraitement, est qu'il n'est pas nécessaire d'expliciter une définition Φ_X de y à partir de X pour montrer qu'elle existe ; il suffit de démontrer que Σ définit implicitement y à partir de X, ce qui est un problème « seulement » coNP-complet [23]. Ceci résulte du théorème suivant, dû à Padoa [26], restreint à la logique propositionnelle et rappelé dans [23].

Théorème 2 *Étant donné $\Sigma \in PROP_{PS}$ et $X \subseteq PS$, soit Σ'_X la formule obtenue en remplaçant de manière uniforme dans Σ toute occurrence de symbole propositionnel x de $Var(\Sigma) \setminus X$ par un nouveau symbole propositionnel x'. Si $y \notin X$ alors Σ définit (de manière implicite) y à partir de X si et seulement si $\Sigma \wedge \Sigma'_X \wedge y \wedge \neg y'$ est incohérent*[2].

2. Trivialement, dans le cas contraire où $y \in X$, Σ définit y à partir de X.

3 Un nouveau prétraitement pour le comptage de modèles

Plutôt que de détecter des portes et de les remplacer dans Σ pour éliminer des variables de l'ensemble des variables de sortie, notre méthode de prétraitement recherche des variables de sortie et les oublie dans Σ. Plus précisément, l'objectif est de déterminer dans un premier temps une bipartition de définition $\langle I, O \rangle$ de Σ.

Définition 3 (bipartition de définition) *Soit Σ une formule propositionnelle de $PROP_{PS}$. Une* bipartition de définition *de Σ est un couple $\langle I, O \rangle$ tel que $I \cup O = Var(\Sigma)$, $I \cap O = \emptyset$, et toute variable $o \in O$ peut être définie à partir de I dans Σ.*

Dans un second temps, des variables de O sont oubliées de la formule Σ pour la simplifier, ce qui donne le prétraitement B + E (*B(ipartition), et É(limination)*) illustré à l'algorithme 1.

Algorithme 1 : B + E

entrée : une formule CNF Σ
sortie : une formule CNF Φ telle que $\|\Phi\| = \|\Sigma\|$

```
1 O←B(Σ);
2 Φ←E(O, Σ);
3 retourner Φ;
```

Un atout important de cet algorithme est que chacune des deux phases peut être adaptée dans le but de garder le prétraitement le plus léger possible du point de vue du temps de calcul. D'un côté, il n'est pas nécessaire de déterminer une bipartition $\langle I, O \rangle$ de Σ telle que le cardinal de O soit maximal (il peut être suffisant de déterminer un ensemble O de cardinal « raisonnable »). D'un autre côté, il n'est pas non plus nécessaire d'oublier dans Σ toutes les variables de O, il peut suffire de se focaliser sur un sous-ensemble $E \subseteq O$. Plus formellement, la correction de B + E est établie par la proposition suivante :

Proposition 1 *Soit Σ une formule propositionnelle de $PROP_{PS}$. Soit $\langle I, O \rangle$ une bipartition de définition de $Var(\Sigma)$. Soit $E \subseteq O$. Alors $\|\Sigma\| = \|\exists E.\Sigma\|$.*

Preuve: Commençons par les deux observations suivantes :

— Soit $E = \{y_1, \ldots, y_m\}$ un sous-ensemble de O. Puisque chaque y_i ($i \in 1, \ldots, m$) est défini à partir de I dans Σ, il existe une formule $\Phi_I^{y_i}$ sur I telle que $\Sigma \models y_i \leftrightarrow \Phi_I^{y_i}$ (*i.e.*, une définition de y_i sur I dans Σ). On note $\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i \in 1,\ldots,m}$ la formule obtenue en remplaçant dans Σ les occurrences de y_i par $\Phi_I^{y_i}$.

— Soit γ_I un terme canonique sur I. Si $\gamma_I \wedge \Sigma$ est cohérent alors il existe un unique modèle ω_{γ_I} de Σ sur $Var(\Sigma)$ qui est modèle de γ_I. De ce fait, tout modèle de Σ est totalement caractérisé par sa restriction sur I, de telle sorte que $\|\Sigma\|$ est égal au nombre de termes canoniques γ_I sur I tel que $\gamma_I \wedge \Sigma$ est cohérent.

On a par construction

$$\Sigma \equiv \bigwedge_{i=1}^{m} (y_i \leftrightarrow \Phi_I^{y_i}) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}.$$

De ce fait, pour tout terme canonique γ_I sur I, $\gamma_I \wedge \Sigma$ est équivalent à

$$\gamma_I \wedge \bigwedge_{i=1}^{m} (y_i \leftrightarrow \Phi_I^{y_i}) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}.$$

Puisque γ_I est un terme canonique sur I et $Var(\Phi_I^{y_i}) \subseteq I$ pour tout $i \in 1,\ldots,m$, on obtient que $\gamma_I \wedge \Phi_I^{y_i}$ est cohérent si et seulement si $\gamma_I \models \Phi_I^{y_i}$; en conséquence, $\gamma_I \wedge \bigwedge_{i=1}^m (y_i \leftrightarrow \Phi_I^{y_i})$ est équivalent à $\gamma_I \wedge \bigwedge_{i=1}^m y_i^*$ où y_i^* $(i \in 1,\ldots,m)$ est y_i quand $\gamma_I \models \Phi_I^{y_i}$ et est $\neg y_i$ dans le cas contraire.

Il s'ensuit que $\gamma_I \wedge \bigwedge_{i=1}^m (y_i \leftrightarrow \Phi_I^{y_i}) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}$ est équivalent à

$$\gamma_I \wedge (\bigwedge_{i=1}^{m} y_i^*) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}.$$

De plus, puisque $Var(\gamma_I \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}) \subseteq I$, $Var(\bigwedge_{i=1}^m y_i^*) \subseteq O$ et $I \cap O = \emptyset$, $\gamma_I \wedge \Sigma$ est cohérent si et seulement si $\gamma_I \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}$ est cohérent. Cependant, comme γ_I est un terme canonique sur I et $Var(\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}) \subseteq I$, c'est précisément le cas lorsque $\gamma_I \models \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}$.

On obtient de ce fait que le nombre de termes canoniques γ_I sur I tel que $\gamma_I \wedge \Sigma$ est cohérent est égal au nombre de modèles de $\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}$ sur I. Autrement dit, on a $\|\Sigma\| = \|\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}\|$.

Puisque $Var(\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}) \cap E = \emptyset$, on a $\exists E.\Sigma \equiv \exists E.(\bigwedge_{i=1}^m (y_i \leftrightarrow \Phi_I^{y_i}) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m})$ $\equiv (\exists E.(\bigwedge_{i=1}^m (y_i \leftrightarrow \Phi_I^{y_i})) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}$.

Finalement, comme $Var(\Phi_I^{y_i}) \cap E = \emptyset$, on a aussi que $\exists E.(\bigwedge_{i=1}^m (y \leftrightarrow \Phi_I^{y_i}))$ est équivalent à $\bigwedge_{i=1}^m (\exists y_i.(y_i \leftrightarrow \Phi_I^{y_i}))$. Comme tout $\exists y_i.(y_i \leftrightarrow \Phi_I^{y_i})$ $(i \in 1,\ldots,m)$ est une formule valide, on obtient que $\exists E.\Sigma \equiv \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}$, ce qui implique que $\|\exists E.\Sigma\| = \|\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}\|$, et finalement que $\|\Sigma\| = \|\exists E.\Sigma\|$.

■

La conjugaison du fait de n'identifier qu'un sous-ensemble O de variables de sortie lors de l'étape de bipartition avec le fait de ne considérer qu'un sous-ensemble $E \subseteq O$ de ces variables lors de la phase d'élimination apporte un réel gain. En effet, le calcul d'une base minimale (*i.e.* un sous-ensemble I de cardinal minimal telle que toute variable de $O = Var(\Sigma) \setminus I$ soit définissable dans Σ à partir de I) [23] serait beaucoup trop coûteux ; en effet, dans le pire des cas, un algorithme de type *branch-and-bound* pour calculer une telle base nécessiterait un nombre de tests de définissabilité exponentiel dans le nombre de variables de Σ. De plus, bien qu'oublier des variables dans Σ diminue évidemment le nombre de variables qui y apparaissent, cela peut aussi conduire à une augmentation exponentielle de sa taille. De ce fait, n'éliminer de Σ qu'un sous-ensemble E des variables de O permet de sélectionner des variables pour lesquelles la phase d'élimination n'augmente pas de manière trop importante la taille de Σ (à la NiVER [32]). Plus précisément, l'élimination d'une variable de sortie ne sera effectuée que si la taille de Σ après l'élimination reste suffisamment faible, une fois l'application de prétraitements supplémentaires achevée. Parmi ces prétraitements additionnels qui préservent l'équivalence, on trouve la simplification d'occurrences et la vivification [28] (déja considérée dans [21]), ayant pour but de réduire certaines clauses, et d'en retirer certaines (pour la vivification).

Exemple 2 (Suite de l'exemple 1) *Aucune équivalence de littéraux, portes ET, OU, ou XOR n'est conséquence logique de* Σ*. Cependant, dans la mesure où* Σ *implique*

$$d \leftrightarrow (a \wedge (b \vee c)) \text{ et } e \leftrightarrow (\neg a \vee (b \leftrightarrow c))$$

$\langle \{a,b,c\}, \{d,e\} \rangle$ *est une bipartition de définition de* $Var(\Sigma)$*. En oubliant* d *et* e *dans* Σ*, les deux clauses non tautologiques* $a \vee c$ *et* $a \vee b \vee c$ *sont générées, ce qui conduit à une formule* `CNF` *équivalente à* $\exists\{d,e\}.\Sigma$ *donnée par :*

$$a \vee b, \quad a \vee c, \quad a \vee b \vee c,$$

qui peut être simplifiée en $(a \vee b) \wedge (a \vee c)$*. Cette formule* `CNF` *admet* 5 *modèles, ce qui est donc aussi le cas de* Σ*.*

L'algorithme 2 montre comment une bipartition $\langle I, O \rangle$ de $Var(\Sigma)$ est calculée de façon gloutonne par `B`. À la ligne 1, `backbone`(Σ) calcule le *backbone* de Σ (l'ensemble des littéraux impliqués par Σ) en utilisant un prouveur SAT, et initialise O avec ces variables (un littéral ℓ appartient au *backbone* de Σ implique que $var(\ell)$ est défini dans Σ à partir de $\emptyset$). La propagation unitaire des contraintes est ensuite appliquée à Σ conjoint aux littéraux de son *backbone*, ce qui conduit à simplifier la formule et à éliminer de celle-ci les variables des littéraux du *backbone*. À la

Algorithme 2 : `B`

entrée : une formule `CNF` Σ
sortie : un ensemble O de variables de sortie, *i.e.*, des variables définies dans Σ à partir de $I = Var(\Sigma) \setminus O$

```
1  ⟨Σ, O⟩← backbone(Σ);
2  V← trier(Var(Σ));
3  I←∅;
4  pour chaque x ∈ V faire
5      si défini?(x, Σ, I ∪ succ(x, V), max#C) alors
6          O←O ∪ {x};
7      sinon
8          I←I ∪ {x};
9  retourner O;
```

ligne 2, on ordonne les variables de Σ de celles possédant le moins d'occurrences de Σ à celles en possédant le plus. À la ligne 5, `défini?` utilise la méthode de Padoa (Théorème 2) afin de déterminer si x est définie dans Σ à partir de $I \cup \texttt{succ}(x, \mathcal{V})$, où $\texttt{succ}(x, \mathcal{V})$ est l'ensemble des variables de V qui apparaissent après x dans V. La fonction `défini?` utilise un prouveur SAT `solve` basé sur l'architecture CDCL pour effectuer le test d'incohérence requis par la méthode de Padoa. Dans notre implémentation, la formule `CNF` d'entrée de `solve` est $\Sigma \wedge \Sigma'_{\emptyset} \wedge \bigwedge_{z \in Var(\Sigma)}((\neg s_z \vee \neg z \vee z') \wedge (\neg s_z \vee z \vee \neg z'))$, à laquelle sont ajoutées des littéraux hypothèses (« *assumptions* ») : pour chaque z appartenant à $I \cup \texttt{succ}(x, \mathcal{V})$, nous introduisons une variable hypothèse s_z, *i.e.* une clause unitaire qui joue un rôle de sélecteur (quand elle est affectée à vrai, elle crée une équivalence entre z et sa copie z') ; x et $\neg x'$ sont elles-aussi ajoutées en tant qu'hypothèses de façon à pouvoir réduire le test de définissabilité considéré sur x à un test d'incohérence. L'intérêt d'utiliser des hypothèses est grand, dans la mesure où cela permet au prouveur SAT de conserver les clauses qu'il apprend, pour aider à la recherche lors des appels suivants. `défini?` admet un paramètre `max#C`, qui borne le nombre de clauses que le prouveur peut apprendre lors de cet appel ; si ce nombre est atteint alors qu'aucune contradiction n'a pu être détectée, la fonction retourne *false* (x n'est pas considérée comme définie dans Σ à partir de $I \cup \texttt{succ}(x, V)$, alors qu'on aurait pu potentiellement démontrer l'inverse avec une valeur de `max#C` supérieure). On voit de ce fait que le nombre de variables de sortie déterminé par `B` n'est pas nécessairement maximal ; ceci est réalisé dans une optique d'efficacité. Enfin, on peut observer que le nombre d'appels à `solve` n'excède pas le nombre de variables de Σ.

L'algorithme 3 montre comment les variables de O sont éliminées de Σ par `E`. P est l'ensemble des variables de E dont l'élimination est possiblement reportée, et est initialisé à la ligne 2 avec l'ensemble O. La boucle principale à la ligne 3 est répétée tant que l'élimination d'une des variables a été réalisée (ligne 16). À la ligne 4, l'ensemble E des variables à tenter d'éliminer durant l'itération courante est initialisé avec P, qui est réinitialisé à $\emptyset$. À la ligne 5, les clauses de Φ sont successivement vivifiées, en utilisant une légère variante de l'algorithme de [21] ; le paramètre additionnel E est utilisé pour ordonner les clauses de Σ de sorte que les littéraux portant sur les variables de E soient traités en premier (c'est-à-dire qu'on va tenter d'éliminer prioritairement les occurrences des littéraux de E). On entre à la ligne 6 dans la boucle interne qui va procéder à une itération par variable de E. Une des variables x de cet ensemble est sélectionnée à la ligne 7 en comptant le nombre $\#(\Phi_x)$ (resp. $\#(\Phi_{\neg x})$) de clauses où x apparaît sous la forme de son littéral positif (resp. négatif) ; la variable x sélectionnée est l'une de celles qui minimisent $\#(\Phi_x) \times \#(\Phi_{\neg x})$, valeur qui majore le nombre de résolvantes possiblement générées par l'élimination de x dans Φ. Puis x est finalement retirée de E à la ligne 8. Ensuite, avant de générer l'ensemble R des résolvantes non tautologiques des clauses de Φ sur x (calculées ligne 13, par la fonction `Res`), on tente dans un premier temps

Algorithme 3 : `E`

entrée : une formule `CNF` Σ et un ensemble de variables de sortie $O \subseteq Var(\Sigma)$
sortie : une formule `CNF` Φ telle que $\Phi \equiv \exists E.\Sigma$ avec $E \subseteq O$

```
1  Φ←Σ;
2  itérer←true ; P←O;
3  tant que itérer faire
4      E←P ; P←∅ ; itérer←false;
5      Φ←vivificationSimpl(Φ, E);
6      tant que E ≠ ∅ faire
7          x←select(E, Φ);
8          E←E \ {x};
9          Φ←occurrenceSimpl(Φ, x);
10         si #(Φ_x) × #(Φ_¬x) > max#Res alors
11             P←P ∪ {x};
12         sinon
13             R←removeSub(Res(x, Φ), Φ);
14             si #((Φ \ Φ_{x,¬x}) ∪ R) ≤ #(Φ) alors
15                 Φ←(Φ \ Φ_{x,¬x}) ∪ R;
16                 itérer←true;
17             sinon
18                 P←P ∪ {x};
19 retourner Φ ;
```

d'éliminer des occurrences de x en utilisant la fonction `occurrenceSimpl` (ligne 9) dans le but de diminuer le cardinal de R. `occurrenceSimpl` est une restriction de l'algorithme de simplification d'occurrences présenté dans [21], où l'on considère les littéraux de $\mathcal{L} = \{x, \neg x\}$ plutôt que l'ensemble des littéraux de Φ. On vérifie ensuite à la ligne 10 que le majorant du cardinal de R recalculé après simplification de Φ ne dépasse pas une constante prédéfinie `max#Res`. Si tel est le cas, alors l'élimination de x dans Φ est possiblement remise à une itération future de la boucle principale, ce qui est réalisé en la réintégrant à l'ensemble P (ligne 11). Dans le cas contraire, l'ensemble R est simplifié en lui retirant les clauses sous-sommées par une autre clause de R ou une clause de Φ (`removeSub`, ligne 13). On vérifie ensuite, à la ligne 14, que l'élimination de x dans Φ (en retirant de Φ les clauses mentionnant x et en lui ajoutant les résolvantes de R) n'augmente pas le nombre de clauses de Φ. S'il y a augmentation alors l'élimination de x dans Φ est possiblement reportée (ligne 18), sinon l'élimination de x dans Φ est actée. On voit qu'il n'y a aucune assurance que toutes les variables de O soient effectivement éliminées par `E`; encore une fois, ce comportement est dicté par un souci d'efficacité.

4 Expérimentations

Nos expérimentations ont été conduites sur 703 instances `CNF` de la SAT LIBrary [3]. Ces instances sont organisées en 8 groupes : BN (*Bayesian networks*) (192), BMC (*Bounded Model Checking*) (18), *Circuit* (41), *Configuration* (35), *Handmade* (58), *Planning* (248), *Random* (104), Qif (7) (*Quantitative Information Flow analysis - security*). Les ordinateurs utilisés pour nos expérimentations étaient équipés de processeurs Intel Xeon E5-2643 (3.30 GHz) avec 32 Gio de RAM et d'un système Linux CentOS. Une limite de temps calcul de 1h et une limite d'espace mémoire de 7.6 Gio ont été prises en compte pour chacune des instances. `max#Res` a été fixé à 500.

Nous avons comparé $\mathtt{B}+\mathtt{E}$ au préprocesseur `pmc` [21], disponible à l'adresse `www.cril.fr/KC/`. Plus précisément, `pmc` a été lancé avec la combinaison de prétraitements $\#eq$, qui allie le calcul du *backbone*, la simplification des occurrences, la vivification des clauses et le remplacement de portes logiques; cette combinaison s'est, en effet, montrée particulièrement efficace comme prétraitement pour le comptage de modèles [21].

Nous avons évalué l'impact de $\mathtt{B}+\mathtt{E}$ (pour plusieurs valeurs de `max#C`) en l'associant à des compteurs exacts de modèles de l'état de l'art. Nous avons notamment évalué son influence sur les logiciels `Cachet` [4] [30] et `sharpSAT` [5] [33], dédiés à ce type de calcul, avec leurs paramètres par défaut. Bien que les approches basées sur la compilation de connaissances effectuent un travail bien plus important que le comptage de modèles (elles calculent une formule Σ^* logiquement équivalente à la formule `CNF` d'entrée Σ, et pas seulement un nombre de modèles), certaines d'entre elles sont en pratique compétitives pour le comptage de modèles. Nous avons donc ajouté à notre étude les résultats obtenus en utilisant le compilateur `C2D` [6] [10, 11] comme compteur de modèles. `C2D` calcule une formule sous forme `Decision-DNNF` Σ^* équivalente à Σ. Dans le pire des cas, la taille de Σ^* est exponentielle dans la taille de Σ; cependant, le nombre de modèles de Σ conditionné par n'importe quel terme cohérent γ peut être calculé efficacement à partir de Σ^*, ce qui permet notamment d'obtenir le nombre de modèles de Σ quand $\gamma = \top$. `C2D` a été lancé avec les paramètres `-count -in_memory -smooth_all`, qui sont ceux à employer lorsque `C2D` est utilisé pour le comptage de modèles.

Dans le cas général, $\mathtt{B}+\mathtt{E}$ ne peut être utilisé comme prétraitement d'un processus de compilation lorsqu'il s'agit par la suite de considérer le conditionnement de la formule initiale par des termes γ cohérents *quelconques*. En effet, lorsque $\mathtt{B}+\mathtt{E}(\Sigma)$ n'est pas équivalente à Σ, la représentation en `Decision-DNNF` $(\mathtt{B}+\mathtt{E}(\Sigma))^*$ calculée par `C2D` n'est pas équivalente à Σ^*. Cela a pour effet de faire perdre la possibilité de calculer efficacement le nombre de modèles après conditionnement pour des termes quelconques; il faut alors restreindre les variables utilisables dans les termes γ à celles apparaissant dans l'ensemble des variables I. Il existe cependant des scénarios où cette forme restreinte de conditionnement peut être suffisante, par exemple quand l'ensemble des variables de Σ peut être partitionné en deux ensembles, le premier étant celui des variables contrôlables (qui pourront être conditionnées) et le deuxième celui des variables non contrôlables restantes. Dans ce cas, on pourrait employer une version modifiée de $\mathtt{B}+\mathtt{E}$ qui assurerait que les variables contrôlables soient insérées dans l'ensemble I dans le but de simplifier la formule Σ avant de la compiler.

La table 1 répertorie le nombre d'instances, parmi les 703 jeux d'essai considérés, qui ont été résolues modulo les ressources (temps et espace) allouées, par les compteurs `Cachet`, `sharpSAT`, et `C2D` (première colonne), quand aucun prétraitement n'est employé (deuxième colonne), quand `pmc` (avec $\#eq$) a été appliqué d'abord (troisième colonne), et finalement quand

3. `www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html`

4. `www.cs.rochester.edu/~kautz/Cachet/`
5. `sites.google.com/site/marcthurley/sharpsat`
6. `reasoning.cs.ucla.edu/c2d/`

compteur	$\emptyset$	pmc	10	100	1000	∞
Cachet	525	558	586	588	594	**602**
sharpSAT	507	537	575	581	586	**593**
C2D	547	602	605	613	616	**621**

TABLE 1 – Nombre d'instances résolues avec les ressources allouées selon les prétraitements $\emptyset$, pmc et B + E.

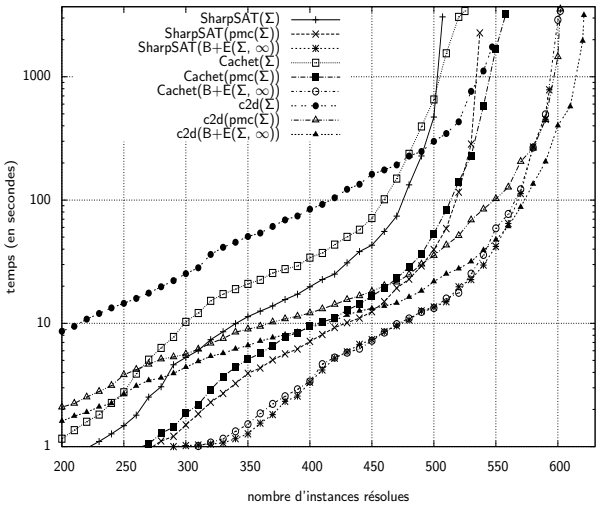


FIGURE 1 – Nombre d'instances résolues par Cachet, sharpSAT et C2D pour les prétraitements $\emptyset$, pmc et B + E. Un point (x, y) indique que le compilateur concerné a résolu x instances avec un temps limite inférieur ou égal à y secondes pour chacune d'entre elles.

B + E(Σ) a été appliqué d'abord, avec différentes valeurs pour le paramètre max#C (colonnes restantes). Le temps de prétraitement est pris en compte dans la limite des 1h pour résoudre l'instance [7].

Les résultats présentés à la table 1montrent les bénéfices apportés par l'application de B + E avant l'appel à un compteur de modèles, mais aussi le gain de performances qu'il apporte en comparaison à pmc. Puisque les meilleures performances de B + E sont atteintes lorsque le paramètre max#C est fixé à ∞, nous avons considéré ce réglage pour les expérimentations suivantes.

Les courbes en cactus données à la figure 1 montrent les performances de Cachet, sharpSAT et C2D, avec et sans les prétraitements pmc et B + E. Pour chaque valeur t sur l'axe des ordonnées (un temps, en seconde, pour compter les modèles) et chaque point sur la courbe pour laquelle cette valeur est atteinte sur l'axe des ordonnées, la valeur correspondante sur l'axe des abscisses donne le nombre d'instances résolues par le compteur pour une limite de temps de t (qui inclut le temps de prétraitement, le cas échéant). Pour des raisons de lisibilité, seuls 10% des points ont été affichés. Encore une fois, on observe que l'utilisation de B + E conduit à de meilleurs résultats que l'utilisation de pmc.

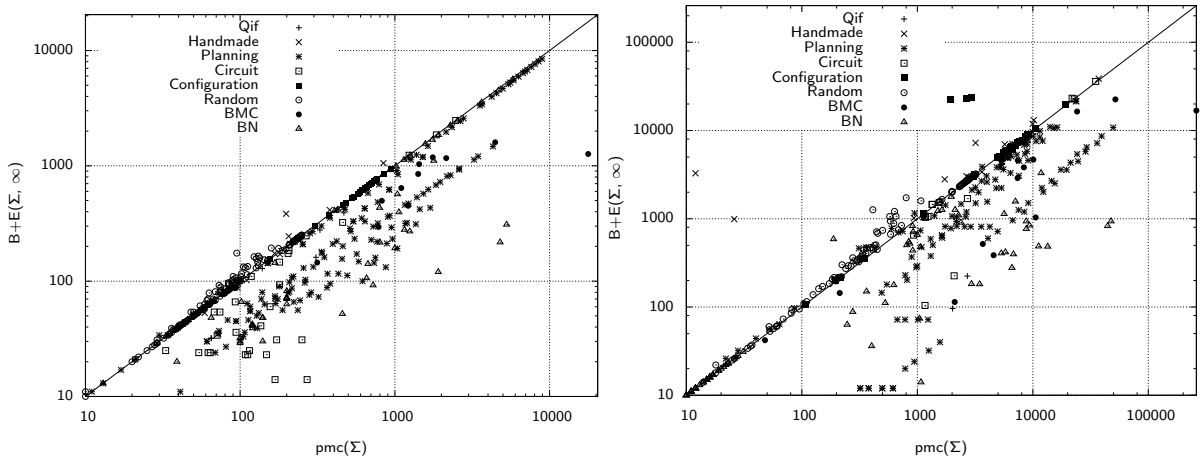
Afin de déterminer la réduction de taille de la formule Σ apportée par l'application de B + E, nous avons considéré deux mesures : $\#var(\Sigma)$, le nombre de variables de Σ, et $\#lit(\Sigma)$, le nombre de littéraux dans Σ (*i.e.*, sa taille) [8].

Les résultats empiriques sont présentés sur les deux diagrammes de dispersion à échelles logarithmiques (a) et (b) de la figure 2 ; dans ces diagrammes, chaque point représente une instance Σ, son abscisse correspondant à la mesure ($\#var$ (a) ou $\#lit$ (b)) pour pmc(Σ) avec $\#eq$, son ordonnée correspondant à la même mesure pour B + E(Σ) (avec max#Conflicts = ∞). Il apparaît clairement que B + E est la plupart du temps meilleur que pmc pour les deux mesures, en particulier pour les instances des familles *Planning*, BMC, et dans une moindre mesure BN.

Finalement, nous avons évalué les gains en terme de temps de calcul apportés à Cachet et C2D (les deux compteurs donnant les meilleurs résultats dans les expérimentations précédentes) grâce à la phase de prétraitement. Ces résultats sont présentés par les deux diagrammes de dispersion (a) et (b), à échelles logarithmiques, de la figure 3. Chaque point correspond à une instance Σ, son abscisse (resp. ordonnée) étant le temps en secondes requis pour calculer $\|\Sigma\|$ en appelant pmc(Σ) (resp. B + E(Σ) avec max#Conflicts= ∞) puis le compteur de modèles sur la formule CNF simplifiée. Encore une fois, peu importe le compteur de modèles finalement employé, B + E apparaît comme étant un préprocesseur plus efficace que pmc. Les résultats présentés sur le diagramme (b) corroborent ceux de la table 1, qui met en évidence des instances pour lesquelles le nombre de modèles peut être calculé en moins d'une heure lorsque B + E est employé, mais pas lorsque le préprocesseur pmc est utilisé à la place.

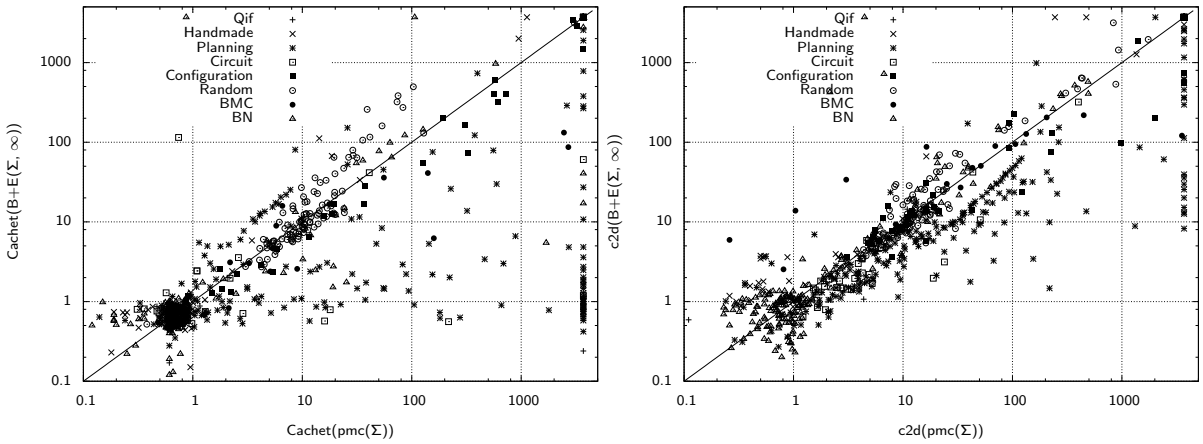
7. Considérer la part du temps de prétraitement dans le temps de calcul global pourrait conduire à biaiser l'interprétation de l'efficacité des processus de prétraitement ; pour certaines instances (voir les résultats détaillés accessibles à partir de www.cril.fr/KC/), le temps relatif de prétraitement peut être conséquent, tout simplement parce que ce processus effectue la quasi-totalité du travail : quand la simplification réalisée de la formule d'entrée Σ est drastique, compter le nombre de modèles de la formule résultante peut devenir trivial.

8. Contrairement à [21], nous n'avons pas calculé la largeur d'arbre du graphe primal associé à Σ, dans la mesure où nous n'avons pas trouvé de logiciel capable de la calculer en un temps raisonnable vu la taille des instances considérées (ce problème est NP-difficile).



(a) *#var* (b) *#lit*

FIGURE 2 – Comparaison du nombre de variables et du nombre de littéraux de $\mathtt{B}+\mathtt{E}(\Sigma)$ et de $\mathtt{pmc}(\Sigma)$. Un point (x, y) correspond à une instance Σ pour laquelle la mesure vaut x pour $\mathtt{pmc}(\Sigma)$ et y pour $\mathtt{B}+\mathtt{E}(\Sigma)$.



(a) $\mathtt{B}+\mathtt{E}$+Cachet vs. pmc+Cachet (b) $\mathtt{B}+\mathtt{E}$+C2D vs. pmc+C2D

FIGURE 3 – Comparaison des performances en terme de temps de calcul pour déterminer le nombre de modèles des approches $\mathtt{B}+\mathtt{E}$+Cachet et pmc+Cachet, puis $\mathtt{B}+\mathtt{E}$+C2D et pmc+C2D. Un point (x, y) correspond à une instance Σ, pour laquelle le temps nécessaire au calcul du nombre de modèles est égal à x secondes quand pmc est le préprocesseur utilisé et y secondes quand il s'agit de $\mathtt{B}+\mathtt{E}$.

5 Conclusion

Nous avons défini un nouveau préprocesseur $\mathtt{B}+\mathtt{E}$, qui associe à une formule CNF Σ donnée une formule CNF $\mathtt{B}+\mathtt{E}(\Sigma)$ qui admet le même nombre de modèles que Σ, mais qui est souvent plus simple au regard de son nombre de variables et de sa taille. $\mathtt{B}+\mathtt{E}$ s'appuie sur la recherche de variables définies dans Σ pour simplifier cette formule. Nous avons expérimenté $\mathtt{B}+\mathtt{E}$ sur de nombreuses instances provenant de différentes familles de jeux d'essai. Nos expérimentations montrent que l'application du prétraitement $\mathtt{B}+\mathtt{E}$ à Σ permet un gain important en terme de temps de calcul pour le comptage de modèles via des compteurs exacts de l'état de l'art, comparé au temps nécessaires à ces compteurs lorsqu'aucun prétraitement n'est appliqué ou lorsque le préprocesseur pmc est employé.

Ce travail ouvre plusieurs perspectives pour des recherches futures. Tout d'abord, en ce qui concerne $\mathtt{B}+\mathtt{E}$, il serait intéressant de considérer d'autres heuristiques dans B pour calculer la bipartition, de déterminer comment adapter la valeur des constantes max#C et max#Res en fonction de l'instance considérée, our

encore de combiner les préprocesseurs B + E et pmc.

D'autres perspectives concernent la notion de comptage de modèles projetés [3], dont le but est le calcul de $\|\exists E.\Sigma\|$, pour un ensemble E et une formule Σ donnés. Un tel calcul est utile, par exemple pour l'analyse et la quantification de flots d'information dans les programmes et des compteurs de modèles projetés ont ainsi été développés à cet effet [20]. Au lieu d'appliquer B + E suivi d'un appel à un compteur de modèles pour calculer $\|\Sigma\|$, on pourrait appliquer B puis invoquer un compteur de modèles projetés (où la projection se ferait sur I). Réciproquement, lorsqu'un comptage de modèles projetés est requis, on pourrait appliquer E sur E et Σ comme un préprocesseur du compteur de modèles projetés considéré. Il serait donc intéressant d'implémenter ces deux approches afin d'évaluer leur intérêt pratique.

Remerciements

Nous adressons nos remerciements aux relecteurs anonymes pour leur lecture attentive, ainsi que leurs commentaires et conseils.

Références

[1] U. Apsel and R. I. Brafman. Lifted MEU by weighted model counting. In *Proc. of AAAI'12*, 2012.

[2] G. Audemard and L. Simon. Predicting learnt clauses quality in modern sat solver. In *Proc. of IJCAI'09*, pages 399–404, 2009.

[3] R. A. Aziz, G. Chu, Ch. J. Muise, and P. J. Stuckey. #∃sat : Projected model counting. In *Proc. of SAT'15*, pages 121–137, 2015.

[4] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #sat and Bayesian inference. In *Proc. of FOCS'03*, pages 340–351, 2003.

[5] F. Bacchus and J. Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Proc. of SAT'04*, pages 341–355, 2004.

[6] E.W. Beth. On Padoa's method in the theory of definition. *Indagationes mathematicae*, 15 :330–339, 1953.

[7] A. Biere. Lingeling essentials, A tutorial on design and implementation aspects of the the SAT solver lingeling. In *Proc. of POS'14*, page 88, 2014.

[8] M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7) :772–799, 2008.

[9] A. Choi, D. Kisa, and A. Darwiche. Compiling probabilistic graphical models using sentential decision diagrams. In *Proc. of ECSQARU'13*, pages 121–132, 2013.

[10] A. Darwiche. Decomposable negation normal form. *Journal of the Association for Computing Machinery*, 48(4) :608–647, 2001.

[11] A. Darwiche. New advances in compiling cnf into decomposable negation normal form. In *Proc. of ECAI'04*, pages 328–332, 2004.

[12] C. Domshlak and J. Hoffmann. Fast probabilistic planning through weighted model counting. In *Proc. of ICAPS'06*, pages 243–252, 2006.

[13] N. Een and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proc. of SAT'05*, pages 61–75, 2005.

[14] L. Feiten, M. Sauer, T. Schubert, A. Czutro, E. Böhl, I. Polian, and B. Becker. #SAT-based vulnerability analysis of security components - A case study. In *Proc. of DFT'12*, pages 49–54, 2012.

[15] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. In *Handbook of Satisfiability*, pages 633–654. 2009.

[16] H. Han and F. Somenzi. Alembic : An efficient algorithm for cnf preprocessing. In *Proc. of DAC'07*, pages 582–587, 2007.

[17] M. Heule, M. Järvisalo, and A. Biere. Clause elimination procedures for cnf formulas. In *Proc. of LPAR'10*, pages 357–371, 2010.

[18] M. Heule, M. Järvisalo, and A. Biere. Efficient cnf simplification based on binary implication graphs. In *Proc. of SAT'11*, pages 201–215, 2011.

[19] M. Järvisalo, A. Biere, and M. Heule. Simulating circuit-level simplifications on cnf. *Journal of Automated Reasoning*, 49(4) :583–619, 2012.

[20] V. Klebanov, N. Manthey, and Ch. J. Muise. Sat-based analysis and quantification of information flow in programs. In *Proc. of QUEST'13*, pages 177–192, 2013.

[21] J.-M. Lagniez and P. Marquis. Preprocessing for propositional model counting. In *Proc. of AAAI'14*, pages 2688–2694, 2014.

[22] J. Lang, P. Liberatore, and P. Marquis. Propositional independence : Formula-variable independence and forgetting. *Journal of Artificial Intelligence Research*, 18 :391–443, 2003.

[23] J. Lang and P. Marquis. On propositional definability. *Artificial Intelligence*, 172(8-9) :991–1017, 2008.

[24] N. Manthey. Coprocessor 2.0 - A flexible CNF simplifier - (tool presentation). In *Proc. of SAT'12*, pages 436–441, 2012.

[25] N. Manthey. Solver description of RISS 2.0 and PRISS 2.0. Technical report, TU Dresden, Knowledge Representation and Reasoning, 2012.

[26] A. Padoa. Essai d'une théorie algébrique des nombres entiers, précédé d'une introduction logique à une théorie déductive quelconque. In *Bibliothèque du Congrès International de Philosophie*, pages 309–365. Paris, 1903.

[27] H. Palacios, B. Bonet, A. Darwiche, and H. Geffner. Pruning conformant plans by counting models on compiled d-DNNF representations. In *Proc. of ICAPS'05*, pages 141–150, 2005.

[28] C. Piette, Y. Hamadi, and L. Saïs. Vivifying propositional clausal formulae. In *Proc. of ECAI'08*, pages 525–529, 2008.

[29] M. Samer and S. Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1) :50–64, 2010.

[30] T. Sang, F. Bacchus, P. Beame, H.A. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *Proc. of SAT'04*, 2004.

[31] T. Sang, P. Beame, and H. A. Kautz. Performing Bayesian inference by weighted model counting. In *Proc. of AAAI'05*, pages 475–482, 2005.

[32] S. Subbarayan and D.K. Pradhan. NiVER : Non increasing variable elimination resolution for preprocessing SAT instances. In *Proc. of SAT'04*, pages 276–291, 2004.

[33] M. Thurley. sharpSAT - counting models with advanced component caching and implicit BCP. In *Proc. of SAT'06*, pages 424–429, 2006.

[34] G.S. Tseitin. *On the complexity of derivation in propositional calculus*, chapter Structures in Constructive Mathematics and Mathematical Logic, pages 115–125. Steklov Mathematical Institute, 1968.

F-CPminer : Une approche pour la localisation de fautes basée sur l'extraction de motifs ensemblistes sous contraintes *

Mehdi Maamar[1,2] **Nadjib Lazaar**[2] **Samir Loudni**[3] **Yahia Lebbah**[1]

[1] Laboratoire LITIO, Université d'Oran 1, Algérie
[2] Laboratoire LIRMM , Université de Montpellier 2, France
[3] Laboratoire GREYC, Université de Caen, France
{maamar,lazaar}@lirmm.fr samir.loudni@unicaen.fr lebbah.yahia@univ-oran.dz

Résumé

Dans cet article nous proposons une approche basée sur la fouille de motifs ensemblistes sous contraintes pour la localisation de fautes dans les programmes, qui est un des processus les plus complexes du test logiciel. Nous formalisons le problème de localisation des fautes comme étant un problème d'extraction des k meilleurs motifs satisfaisant un ensemble de contraintes modélisant les instructions les plus suspectes. Nous faisons appel à la Programmation Par Contraintes (PPC) pour modéliser et résoudre le problème de localisation. L'approche est basée sur deux étapes : i) Extraction des top-k suites d'instructions les plus suspectes ; ii) Localisation des fautes par post-traitement des top-k motifs. Les expérimentations menées sur une série de programmes montrent que notre approche offre une localisation plus précise comparée à une approche standard.

Abstract

We introduce in this paper an itemset mining approach to tackle the fault localization problem, which is one of the most difficult processes in software debugging. We formalize the problem of fault localisation as finding the k best patterns satisfying a set of constraints modelling the most suspicious statements. We use a Constraint Programming (CP) approach to model and to solve our itemset based fault localization problem. Our approach consists of two steps : i) mining top-k suspicious suites of statements ; ii) fault localization by processing top-k patterns. Experiments performed on standard benchmark programs show that our approach enables to propose a more precise localization than a standard approach.

*Papier publié en version longue dans le journal Automated Software Engineering ASE'16 [14].

1 Introduction

L'aide à la localisation de fautes dans les programmes à partir de traces d'exécution est une question cruciale lors de la mise au point et du test de logiciels. En effet, lorsqu'un programme contient des erreurs, un model-checker fournit une trace d'exécution souvent longue et difficile à analyser, et de ce fait d'un intérêt limité pour le programmeur qui doit débugger son programme. La localisation des portions de code qui contiennent des erreurs est donc souvent un processus difficile et coûteux, même pour des programmeurs expérimentés. Au cours de la dernière décennie, plusieurs techniques automatisées ont été proposées pour répondre à ce problème.

La plupart de ces techniques comparent deux types de traces d'exécution, les exécutions correctes (*positives*) et les exécutions erronées (*negatives*) [10, 9]. Ces méthodes se basent sur une fonction de score pour évaluer le caractère suspect de chaque instruction dans le programme en exploitant les occurrences des instructions dans les traces négatives et positives. L'intuition sous-jacente est que les instructions ayant les scores les plus élevés sont les plus suspectes pour contenir des fautes, ce qui permet d'ordonner les instructions d'un programme de la plus suspecte à la plus innocente. Toutefois, l'inconvénient majeur de ces techniques est que les dépendances entre les instructions sont ignorées.

En outre, ces dernières années, le problème de localisation de fautes a été traité avec des techniques de fouille de données. Cellier et al. [1, 2] proposent une approche basée sur les règles d'associations et l'analyse

formelle de concepts (FCA) pour assister la localisation de fautes. Nessa et al. [15] proposent de générer des sous-séquences d'instructions de longueur N, appelées N-grams, à partir des traces.

D'autre part, plusieurs travaux en fouille de données ont mis en avant l'intérêt d'utiliser les contraintes pour indiquer le type de motifs à extraire afin de cibler le processus d'extraction suivant les centres d'intérêt de l'utilisateur. Comme le procédé produit en général un nombre important de motifs, un grand effort est fait pour mieux comprendre l'information fragmentée véhiculée par les motifs et produire des *sous-ensembles de motifs* satisfaisant des propriétés sur l'ensemble de tous les motifs [5, 19]. L'extraction des top-k motifs est une voie récente en fouille de données sous contraintes permettant de produire des ensembles de motifs intéressants [3].

Dans ce papier, nous formalisons le problème de localisation de fautes comme un problème d'extraction des k meilleurs motifs satisfaisant un ensemble de contraintes modélisant les instructions les plus suspectes. Nous utilisons la couverture des cas de test d'un programme collectée durant la phase de test. Notre approche qui bénéficie des travaux récents sur la fertilisation croisée entre la fouille de données et la programmation par contraintes [7, 11], est réalisée en deux étapes : i) extraction des top-k suites d'instructions les plus suspectes selon une relation de dominance et ii) classement des instructions issues des top-k motifs pour localiser la faute.

Les expérimentations menées sur une série de programmes (*Siemens suite*) montrent que notre approche permet de proposer une localisation plus précise comparée à la technique de localisation de fautes la plus populaire Tarantula [9].

L'article est organisé comme suit. La section 2 introduit les concepts de base. La section 3 présente les notions principales liées à l'extraction de motifs. La section 4 détaille notre approche pour la localisation de fautes. Les résultats des expérimentations sont donnés dans la section 5. Finalement, nous discuterons nos expérimentations en concluant avec des perspectives.

2 Contexte et définitions

Cette section introduit les notions et les définitions habituellement utilisées en localisation de fautes et en satisfaction de contraintes.

2.1 Problème de localisation de fautes

En génie logiciel, une défaillance est une déviation entre le résultat attendu et le résultat obtenu. Une erreur est une partie du programme qui est susceptible de conduire à une défaillance. Une faute est la cause supposée ou adjugée d'une erreur [12]. L'objectif de la localisation de fautes est d'identifier la cause principale des symptômes observés dans les cas de test.

Soit un programme avec faute P ayant n lignes $L = \{e_1, e_2, ..., e_n\}$. La figure 1 montre un exemple de programme "Compteur de caractères", où nous avons $L = \{e_1, e_2, ..., e_{10}\}$.

Un cas de test tc_i est un tuple $\langle D_i, O_i \rangle$, où D_i est l'ensemble des paramètres d'entrée pour déterminer si un programme P se comporte comme prévu ou non, et O_i est la sortie attendue du programme.

Soit $\langle D_i, O_i \rangle$ un cas de test et A_i la sortie courante retournée par un programme P suite à l'exécution de l'entrée D_i. Si $A_i = O_i$, alors le cas de test tc_i est considéré comme correct (c.-à-d. positif), erroné (c.-à-d. négatif) dans le cas contraire.

Une suite de test $T = \{tc_1, tc_2, ..., tc_m\}$ est l'ensemble des m cas de test destinés à tester si le programme P respecte l'ensemble des exigences spécifiées.

Étant donné un cas de test tc_i et un programme P, l'ensemble des instructions de P exécutées (au moins une fois) avec tc_i désigne la couverture du cas de test $I_i = (I_{i,1}, ..., I_{i,n})$, où $I_{i,j} = 1$ si l'instruction à la ligne j est exécutée, 0 sinon. La couverture d'un cas de test indique quelles sont les parties actives du programme durant une exécution spécifique.

Par exemple, le cas de test tc_4 dans la figure 1 couvre les instructions ($e_1, e_2, e_3, e_4, e_6, e_7, e_{10}$). Le vecteur de couverture correspondant est $I_4 = (1, 1, 1, 1, 0, 1, 1, 0, 0, 1)$.

2.2 Problèmes de Satisfaction de Contraintes

Un CSP est un triplet $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ tel que $\mathcal{X} = \{x_1, ..., x_n\}$ est l'ensemble fini des variables. $D = \{D_1, ..., D_n\}$ est l'ensemble des domaines des variables, où chaque D_i est un ensemble fini contenant les valeurs possibles pour la variable x_i. $\mathcal{C} = \{C_1, ..., C_e\}$ est l'ensemble des contraintes. Chaque contrainte C_i exprime une relation sur un sous-ensemble de variables de $\mathcal{X}$ appelé $var(C_i)$. Le but est de trouver une instanciation $(x_i = d_i)$ avec $d_i \in D_i$ pour $i = 1, ..., n$, telle que toutes les contraintes soient satisfaites.

La programmation par contraintes propose des contraintes très expressives. On peut dénoter les contraintes prédéfinies (c.-à-d., les contraintes arithmétiques), contraintes données en extension (liste des combinaisons de valeurs autorisées/interdites), les combinaisons logiques de contraintes. Un autre type de contraintes sont les *contraintes réifiées*, aussi connues comme les *méta contraintes*. Une contrainte réifiée $b \leftrightarrow c$ implique une variable booléenne b et une contrainte c, ceci est équivalent à $(b = 1 \wedge c) \vee (b = 0 \wedge \neg c)$. Ce type de

Programme : Compteur de caractères	Cas de test tc_1	tc_2	tc_3	tc_4	tc_5	tc_6	tc_7	tc_8
`function count (char *s) {`								
`int let, dig, other, i = 0;`								
`char c;`								
e_1 : `while (c = s[i++]) {`	1	1	1	1	1	1	1	1
e_2 : `if('A'<=c && 'Z'>=c)`	1	1	1	1	1	1	0	1
e_3 : `let += 2; //- faute -`	1	1	1	1	1	1	0	0
e_4 : `else if ( 'a'<=c && 'z'>=c )`	1	1	1	1	1	0	0	1
e_5 : `let += 1;`	1	1	0	0	1	0	0	0
e_6 : `else if ( '0'<=c && '9'>=c )`	1	1	1	1	0	0	0	1
e_7 : `dig += 1;`	0	1	0	1	0	0	0	0
e_8 : `else if (isprint (c))`	1	0	1	0	0	0	0	1
e_9 : `other += 1;`	1	0	1	0	0	0	0	1
e_{10} : `printf("%d %d %d\n", let, dig, other);}`	1	1	1	1	1	1	1	1
Positif / Négatif	N	N	N	N	N	N	P	P

FIGURE 1 – Exemple d'un programme avec la matrice de couverture associée

contraintes est utile pour exprimer des formules propositionnelles sur les contraintes, des formules qui sont utilisées en fouille de données (voir la section 3).

3 Extraction de motifs ensemblistes

Cette section présente les notions principales liées à l'extraction de motifs, ainsi que la modélisation sous forme de CSP des problèmes d'extraction de motifs sous contraintes [4, 7].

3.1 Définitions

Soit $\mathcal{I}$ un ensemble de littéraux distincts appelés items et $\mathcal{T} = \{1, ..., m\}$ un ensemble d'identifiants de transactions. Un motif ensembliste d'items est un sous-ensemble non vide de $\mathcal{I}$. Ces motifs sont regroupés dans le langage $\mathcal{L}_I = 2^{\mathcal{I}} \setminus \emptyset$. Un jeu de données transactionnel est l'ensemble $\mathcal{D} \subseteq \mathcal{I} \times \mathcal{T}$.

Définition 1 (Couverture et fréquence) *La couverture d'un motif x est l'ensemble de tous les identifiants de transactions dans lesquelles x apparaît : $cover_{\mathcal{D}}(x) = \{t \in \mathcal{T} | \forall i \in x, (i,t) \in \mathcal{D}\}$.*
La fréquence d'un motif x représente le cardinal de sa couverture $freq(x) = |cover_{\mathcal{D}}(x)|$

L'extraction de motifs sous contraintes consiste à extraire les motifs x de $\mathcal{L}_I$ satisfaisant une requête $q(x)$ (conjonction de contraintes) : $\{x \in \mathcal{L}_{\mathcal{I}} \mid q(x)$ *est vrai*$\}$. La contrainte de motifs fréquents est définie en sélectionnant les motifs dont la fréquence est supérieure à un seuil donné $min_{fr} : freq(x) \geq min_{fr}$. Une autre contrainte est la contrainte de *taille* qui contraint le nombre d'items d'un motif x.

Dans de nombreuses applications, il apparaît très approprié de relever les contrastes entre des sous-ensembles de transactions, tels que les classes de molécules saines et toxiques, les cas de test positifs et négatifs pour la localisation de fautes (voir section 4). Le taux d'émergence est une mesure de contraste très utilisée [16], elle permet d'extraire des motifs dont la fréquence varie fortement d'un jeu de données à un autre.

Une limite bien connue de l'extraction de motifs est que le nombre de motifs produits peut être très grand dû à la redondance que peut contenir une collection de motifs par rapport à une mesure m donnée. En effet, deux motifs x_i et x_j sont équivalents si $m(x_i) = m(x_j)$. Un ensemble de motifs équivalents forme une classe d'équivalence par rapport à m. Le motif le plus grand, au sens de l'inclusion, d'une classe d'équivalence est appelé *motif fermé*.

Définition 2 (Motif fermé) *Un motif $x \in \mathcal{L}_{\mathcal{I}}$ est fermé par rapport à la fréquence ssi $\forall\, y \supsetneq x, freq(y) < freq(x)$.*

Les motifs fermés structurent le treillis des motifs en classes d'équivalence. Tous les motifs appartenant à une même classe d'équivalence ont la même couverture. Les motifs fermés correspondent aux éléments maximaux (au sens de la taille des motifs) des classes d'équivalence.

Par ailleurs, l'utilisateur est souvent intéressé par la découverte de motifs plus riches satisfaisant des contraintes portant sur un ensemble de motifs et non pas sur un seul. Ces contraintes sont définies comme étant des contraintes sur des *ensembles de motifs* [5] ou sur des *motifs n-aires* [11]. L'approche que nous proposons dans ce papier permet de traiter ce type de motifs tel que les top-k motifs.

Définition 3 (top-k motifs) *Soit m une mesure d'intérêt et k un entier. L'ensemble des top-k motifs, est l'ensemble des k meilleurs motifs par rapport à la mesure m :*

$$topk = \{x \in \mathcal{L}_{\mathcal{I}} \mid freq(x) \geq 1 \wedge \nexists y_1, \ldots, y_k \in \mathcal{L}_{\mathcal{I}} : \forall 1 \leq j \leq k, m(y_j) > m(x)\}$$

3.2 Modélisation sous forme d'un CSP

Soit $\mathcal{D}$ un jeu de données où $\mathcal{I}$ est l'ensemble de ses n items et $\mathcal{T}$ l'ensemble de ses m transactions. $\mathcal{D}$

peut être représenté par une matrice (m, n) booléenne d telle que : $\forall t \in \mathcal{T}, \forall i \in \mathcal{I}$, $(d_{t,i} = 1) \Leftrightarrow (i \in t)$

Dans [4], les auteurs modélisent le motif recherché $M \subseteq \mathcal{I}$ et son support en introduisant deux ensembles de variables booléennes : i) variables d'items $\{M_1, M_2, ..., M_n\}$ où $(M_i = 1)$ ssi $(i \in M)$ et ii) variables de transactions $\{T_1, T_2, ..., T_m\}$ où $(T_t = 1)$ ssi $(M \subseteq t)$.

La relation entre le motif recherché M, son support et le jeu de données $\mathcal{D}$ est établie par des contraintes réifiées imposant que, pour chaque transaction t, $(T_t = 1)$ ssi M est un sous ensemble de t :

$$\forall t \in \mathcal{T} : (T_t = 1) \leftrightarrow \sum_{i \in \mathcal{I}} M_i \times (1 - d_{t,i}) = 0 \qquad (1)$$

L'encodage booléen permet d'exprimer de façon simple certaines mesures usuelles telles que :

- $freq(M) = \sum_{t \in \mathcal{T}} T_t$.
- $taille(M) = \sum_{i \in \mathcal{I}} M_i$.

La contrainte de fréquence minimale $freq(M) \geq min_{fr}$ (où min_{fr} représente un seuil) est encodée par la contrainte : $\sum_{t \in \mathcal{T}} T_t \geq min_{fr}$.
De la même manière, la contrainte de taille minimale $taille(M) \geq \alpha$ (où α représente un seuil) est encodée par la contrainte : $\sum_{i \in \mathcal{I}} M_i \geq \alpha$.

Finalement, la contrainte de fermeture $fermé_m(M)$ $(m = freq)$ qui assure qu'un motif M ne possède aucun sur-ensemble avec la même fréquence, est encodée par l'équation (2).

$$\forall i \in \mathcal{I} : (M_i = 1) \leftrightarrow \sum_{t \in \mathcal{T}} T_t \times (1 - d_{t,i}) = 0 \qquad (2)$$

4 Localisation de fautes par extraction de motifs

Cette section détaille notre modélisation du problème de localisation de fautes comme un problème d'extraction des k motifs les plus suspects et présente l'algorithme pour les générer. Nous décrivons également comment exploiter ces top-k motifs pour retourner à la fin un classement précis des instructions pour localiser la faute.

4.1 Modélisation

Soit $L = \{e_1, ..., e_n\}$ l'ensemble des instructions composant le programme P et $T = \{tc_1, ..., tc_m\}$ l'ensemble des cas de test. La base transactionnelle $\mathcal{D}$ est définie comme suit : (i) chaque instruction de L correspond à un item de $\mathcal{I}$, (ii) la couverture de chaque cas de test tc_i forme une transaction dans $\mathcal{T}$. Par ailleurs, pour découvrir les contrastes entre les sous-ensembles de transactions, $\mathcal{T}$ est partitionné en deux sous-ensembles disjoints $\mathcal{T}^+$ et $\mathcal{T}^-$. $\mathcal{T}^+$ (resp. $\mathcal{T}^-$) représente l'ensemble des couvertures des cas de test positifs (resp. négatifs).

Soit d la matrice booléenne représentant la base transactionnelle $\mathcal{D}$. Nous avons alors, $\forall t \in \mathcal{T}, \forall i \in \mathcal{I}$, $(d_{t,i} = 1)$ si et seulement si l'instruction i est exécutée par le cas de test t. On notera respectivement par d^+ et d^- les matrices booléenes des deux bases transactionnelles positive et négative. Dans l'exemple de la figure 1, la couverture du cas de test tc_5 est $I_5 = (1, 1, 1, 1, 1, 0, 0, 0, 0, 1)$. Comme tc_5 est un cas de test qui a échoué, alors $I_5 \in \mathcal{T}^-$.

Soit M le motif suspect recherché (c.-à-d., une suite d'instructions). Comme dans [4], nous introduisons deux ensembles de variables booléennes : n variables $\{M_1, ..., M_n\}$ pour représenter le motif M et m variables $\{T_1, ..., T_m\}$ pour représenter le support de M. Comme pour $\mathcal{T}$, l'ensemble $\{T_1, ..., T_m\}$ est partitionné en deux sous-ensembles disjoints : les variables $\{T_t^+\}$ représentant le support de M dans $\mathcal{T}^+$ et les variables $\{T_t^-\}$ représentant le support de M dans $\mathcal{T}^-$. Nous définissons aussi la mesure de fréquence dans $\mathcal{T}^+$ (resp. $\mathcal{T}^-$) par $freq^+(M) = \sum_{t \in \mathcal{T}^+} T_t^+$ (resp.$freq^-(M) = \sum_{t \in \mathcal{T}^-} T_t^-$).
Afin de réduire la redondance dans les suites d'instructions extraites, nous imposons la contrainte de fermeture $fermé_{freq}(M)$. Dans notre cas, cette contrainte est imposée sur $\mathcal{T}^+$ et $\mathcal{T}^-$ en même temps, ce qui garantit que M n'a pas de sur-ensemble avec les mêmes fréquences dans les deux ensembles. Ceci est encodé avec l'équation (3), qui est une version décomposée de l'équation (2) sur $\mathcal{T}^+$ et $\mathcal{T}^-$.

$$\forall i \in \mathcal{I} : (M_i = 1) \leftrightarrow \left(\sum_{t \in \mathcal{T}^+} T_t^+ \times (1 - d_{t,i}^+) = 0 \right) \wedge \left(\sum_{t \in \mathcal{T}^-} T_t^- \times (1 - d_{t,i}^-) = 0 \right) \qquad (3)$$

4.2 top-k motifs suspects

L'intuition qui est derrière la grande majorité des méthodes de localisation de fautes est que les instructions qui apparaissent très souvent dans les traces d'exécution négatives sont considérées comme étant les plus suspectes, tandis que les instructions qui apparaissent seulement dans les traces d'exécutions positives sont considérées comme étant les plus innocentes [6, 9, 15, 18]. Pour extraire les motifs les plus suspects, nous définissons une relation de dominance $\succ_{\mathcal{R}}$ entre les motifs en fonction de leur niveau de suspicion.

Définition 4 (Relation de dominance) *Étant donnée une bipartition de $\mathcal{T}$ en deux sous-ensembles*

disjoints $\mathcal{T}^+$ et $\mathcal{T}^-$, un motif M domine un autre motif M' (noté $M \succ_{\mathcal{R}} M'$) ssi :

$$\big[freq^-(M) > freq^-(M')\big] \vee \big[(freq^-(M) = freq^-(M')) \wedge (freq^+(M) < freq^+(M'))\big] \quad (4)$$

La relation de dominance exprime le fait que M sera préféré à M', noté $M \succ_{\mathcal{R}} M'$, ssi M est plus fréquent que M' dans $\mathcal{T}^-$. Si les deux motifs couvrent exactement le même ensemble de transactions dans $\mathcal{T}^-$, alors le motif le moins fréquent dans $\mathcal{T}^+$ sera préféré. Notons que deux motifs M et M' peuvent avoir le même degré de suspicion (noté $M =_{\mathcal{R}} M'$) si :

$$\big[freq^-(M) = freq^-(M')\big] \wedge \big[freq^+(M) = freq^+(M')\big] \quad (5)$$

En exploitant la relation de dominance $\succ_{\mathcal{R}}$, nous proposons la notion de top-k motifs suspects.

Définition 5 (top-k motifs suspects) *Un motif M est dans un top-k motifs suspects (par rapport à $\succ_{\mathcal{R}}$) ssi $\nexists P_1, \dots P_k, \in \mathcal{L}_{\mathcal{I}}, \forall 1 \leq j \leq k,\ P_j \succ_{\mathcal{R}} M$.*

Ainsi, un motif M est dans un top-k motifs suspects s'il n'existe pas plus de $(k-1)$ motifs qui le dominent. Un ensemble de top-k motifs suspects est défini comme suit :

$$\mathcal{S} = \{M \in \mathcal{L}_{\mathcal{I}} \mid \texttt{elementaire}\,(M) \wedge \nexists P_1, \dots P_k, \in \mathcal{L}_{\mathcal{I}}, \forall 1 \leq j \leq k,\ P_j \succ_{\mathcal{R}} M\}$$

La contrainte $\texttt{elementaire}(M)$ permet de préciser que le motif M doit satisfaire une propriété de base. Dans notre cas, nous imposons que le motif recherché doit satisfaire les propriétés suivantes :

$$fermé_{freq}(M) \wedge freq^-(M) \geq 1 \wedge taille(M) \geq 1$$

Ce qui requiert qu'un motif M doit être fermé sur les cas de test positifs et négatifs, ceci permet d'avoir le motif le plus grand pour chaque degré de suspicion. Le motif M doit aussi inclure au moins une instruction et apparaître au moins une fois dans les cas de test négatifs.

4.3 Extraction des top-k motifs les plus suspects

Cette partie montre comment les top-k motifs suspects peuvent être extraits en faisant appel au contraintes postées dynamiquement durant la recherche [19]. L'idée est d'exploiter la relation de dominance (noté $\succ_{\mathcal{R}}$) entre les ensembles d'instructions pour produire un raffinement continu dans les motifs extraits grâce aux contraintes postées dynamiquement durant le processus de fouille. Chaque contrainte dynamique va imposer qu'aucun des motifs suspects déjà extraits n'est meilleur (au sens de la relation $\succ_{\mathcal{R}}$) que le motif suivant qui est recherché. Ce processus s'arrête lorsque aucune solution meilleure ne peut être encore obtenue.

Algorithm 1: Extraction des top-k motifs suspects

```
   Données T+, T−, k
   Sorties S : top-k motifs les plus suspects
1  C ← elementaire(M) ;   S ← ∅ ;   i ← 1 ;
2  répéter
3      P ← SolveNext(C)
4      si P ≠ ∅ alors S.add(P) ;   i ← i + 1
   jusqu'à (i > k) ou (P = ∅) ;
5  Trier S selon l'ordre décroissant ≻R ;
6  tant que P ≠ ∅ faire
7      C.add(M ≻R Sk) ;
8      P ← SolveNext(C) ;
9      si P ≠ ∅ alors
10         S.remove(Sk) ;
11         Insérer P dans S selon l'ordre décroissant ≻R ;
       fin
   fin
   retourner S ;
```

L'algorithme 1 prend comme entrées les cas de tests positifs et négatifs ($\mathcal{T}^+$ et $\mathcal{T}^-$), un entier positif k, et retourne comme sortie les top-k motifs les plus suspects. L'algorithme commence avec un ensemble de contraintes qui est la contrainte $\texttt{elementaire}(M)$ (ligne 1). Il recherche les k premiers motifs suspects qui sont des solutions du système de contraintes courant, en utilisant la fonction $SolveNext(\mathcal{C})$ (lignes 2-4). Durant la recherche, une liste $\mathcal{S}$ des top-k motifs suspects est maintenue. Une fois que les k motifs sont trouvés, ils sont triés dans un ordre décroissant selon $\succ_{\mathcal{R}}$ (ligne 5). A chaque fois qu'un nouveau motif est trouvé, nous supprimons de $\mathcal{S}$ le motif le moins préféré S_k selon $\succ_{\mathcal{R}}$ (ligne 10), nous ajoutons le nouveau motif dans $\mathcal{S}$ dans l'ordre décroissant selon $\succ_{\mathcal{R}}$ (ligne 11) et postons dynamiquement une nouvelle contrainte ($M \succ_{\mathcal{R}} S_k$) à ligne 7, qui garantit que le nouveau motif recherché M doit être meilleur selon la relation $\succ_{\mathcal{R}}$ que le moins préféré des motifs dans la liste $\mathcal{S}$ courante des top-k. Ce processus est répété jusqu'à ce qu'aucun motif ne soit généré.

4.4 Localisation de fautes par post-traitement des top-k motifs suspects

Notre algorithme top-k retourne une liste ordonnée des k meilleurs motifs $\mathcal{S} = \langle \mathcal{S}_1, \dots, \mathcal{S}_k \rangle$. Chaque motif $\mathcal{S}_i$ représente un sous-ensemble d'instructions pouvant expliquer et localiser la faute. D'un motif à un autre, des instructions *apparaissent/disparaissent* en fonction de leurs fréquences dans les bases positives/négatives ($freq^+$, $freq^-$).

Nous proposons dans cette section un algorithme (algorithme.2) qui prend en entrée les top-k motifs les

plus suspects et retourne une liste *Loc* contenant un classement précis des instructions susceptibles de localiser la faute (ligne 14). La liste *Loc* inclut trois listes ordonnées notées Ω_1, Ω_2 et Ω_3. Les éléments de Ω_1 sont classés en premier, suivis des éléments de Ω_2 et enfin Ω_3 qui contient les instructions les moins suspectes.

L'algorithme 2 commence par fusionner les motifs équivalents de $\mathcal{S}$ (c.-à-d., équivalents au sens de la relation $=_{\mathcal{R}}$) à la ligne 1 et renvoie une nouvelle liste $\mathcal{SM}$. Notons que la liste retournée $\mathcal{SM}$ peut être égale à la liste initiale des top-k $\mathcal{S}$ s'il n'existe aucune paire de motifs équivalents. Les listes Ω_1 et Ω_3 sont initialisées à vide, alors que la liste Ω_2 est initialisée avec les instructions qui apparaissent dans $\mathcal{SM}_1$ (le plus suspect des motifs) (ligne 3).

Ensuite, l'algorithme 2 essaye de différencier les instructions de Ω_2 en tirant profit des trois propriétés suivantes 1, 2 et 3.

Propriété 1 *Étant donnés les top-k motifs $\mathcal{S}$, $\mathcal{SM}_1$ est une sur-approximation de la localisation de la faute :* $\forall S_i \in \mathcal{S} : (freq_1^- = freq_i^-) \wedge (freq_1^+ = freq_i^+) \Rightarrow S_i \subseteq \mathcal{SM}_1$

Comme $\mathcal{SM}_1$ correspond au motif le plus suspect, il est très susceptible de contenir l'instruction fautive. Toutefois, il contient aussi d'autres instructions qui ont un certain degré de suspicion. C'est pourquoi, dans l'algorithme 2, Ω_2 est initialisée à $\mathcal{SM}_1$ à la ligne 3.

Propriété 2 *Soit l'ensemble de motifs $\mathcal{SM}$ et une sur-approximation de la localisation de la faute $\mathcal{SM}_1$ (Ω_2), certaines instructions de $\mathcal{SM}_1$ disparaissent dans les motifs suivants* $\mathcal{SM}_i \in \mathcal{SM}(i = 2..|\mathcal{SM}|) : (\mathcal{SM}_1 \setminus \mathcal{SM}_i) \neq \emptyset$.

Dans l'algorithme 2, les instructions qui sont dans Ω_2 et disparaissent dans un certain motif $\mathcal{SM}_i$ sont notées Δ_D (ligne 5). Selon les valeurs de fréquences, nous avons deux cas :

1. $\mathcal{SM}_i$ a la même fréquence que $\mathcal{SM}_1$ dans la base négative mais le motif $\mathcal{SM}_i$ est plus fréquent dans la base positive que le motif $\mathcal{SM}_1$. Ainsi, les instructions de Δ_D sont moins fréquentes dans la base positive comparée à celles de $(\Omega_2 \setminus \Delta_D)$. Ainsi, les instructions de Δ_D sont alors plus suspectes que les instructions restantes dans $(\Omega_2 \setminus \Delta_D)$ et doivent être classées en premier (supprimées de Ω_2 (ligne 8) et ajoutées à Ω_1 (ligne 7)).
2. $\mathcal{SM}_i$ est moins fréquent dans la base négative que $\mathcal{SM}_1$. Dans ce deuxième cas, les instructions de Δ_D doivent aussi être classées en premier et être ajoutées à Ω_1.

Par conséquent, Ω_1 contient les instructions les plus suspectes provenant de l'état initial de Ω_2. Les instructions restantes dans Ω_2 qui apparaissent dans tous les motifs $\mathcal{SM}_i$ sont classées après celles de Ω_1.

Propriété 3 *Étant donné l'ensemble de motifs $\mathcal{SM}$ et une sur-approximation de la localisation de la faute $\mathcal{SM}_1$, certaines instructions apparaissent dans les motifs suivants* $\mathcal{SM}_i \in \mathcal{SM} : (\mathcal{SM}_i \setminus \mathcal{SM}_1) \neq \emptyset$.

Selon la propriété 3, certaines instructions qui ne sont pas dans Ω_2 et apparaissent dans les autres motifs $\mathcal{SM}_i$ (ligne 9) sont notées Δ_A. Tous les éléments de Δ_A qui n'ont pas déjà été classés ni dans Ω_1 ni dans Ω_2 (ligne 12) sont ajoutés à Ω_3 (ligne 13) et classée après Ω_2 comme étant les instructions les moins suspectes (ligne 14).

En résumé, la première liste Ω_1 classe les instructions de $\mathcal{SM}_1$ selon leur ordre de disparition dans $\mathcal{SM}_2..\mathcal{SM}_l$ (ligne 4). La liste Ω_2 contient les instructions restantes de $\mathcal{SM}_1$ qui apparaissent dans tous les motifs $\mathcal{SM}_i(i = 1..k)$. Finalement, la liste Ω_3 contient les instructions qui n'appartiennent pas à $\mathcal{SM}_1$ et qui apparaissent graduellement dans $\mathcal{SM}_2..\mathcal{SM}_l$.

Algorithm 2: Localisation de fautes par les top-k

Données $\mathcal{S} = \langle S_1, \ldots, S_k \rangle$, fréquences de chaque S_i : $(freq_i^+, freq_i^-)$
Sorties une liste ordonnée des instructions $Loc = \langle \Omega_1, \Omega_2, \Omega_3 \rangle$

1 $\mathcal{SM} \leftarrow Fusionner(\mathcal{S})$;
2 $\Omega_1 \leftarrow \langle\rangle$; $\Omega_3 \leftarrow \langle\rangle$; $Loc \leftarrow \langle\rangle$;
3 $\Omega_2 \leftarrow \mathcal{SM}_1$; $l \leftarrow |\mathcal{SM}|$;
4 **pour chaque** $i \in 2..l$ **faire**
5 $\quad \Delta_D \leftarrow \Omega_2 \setminus \mathcal{SM}_i$;
6 $\quad$ **si** $\Delta_D \neq \emptyset$ **alors**
7 $\quad\quad \Omega_1.add(\Delta_D)$;
8 $\quad\quad \Omega_2.removeAll(\Delta_D)$;
$\quad$ **fin**
9 $\quad \Delta_A \leftarrow \mathcal{SM}_i \setminus \mathcal{SM}_{i-1}$;
10 $\quad \omega \leftarrow \emptyset$;
11 $\quad$ **pour chaque** $b \in \Delta_A$ **faire**
12 $\quad\quad$ **si** $(\forall \omega' \in \Omega_3, \forall \omega" \in \Omega_1 : b \notin \omega' \wedge b \notin \omega")$ **alors** $\omega \leftarrow \omega \cup \{b\}$
$\quad$ **fin**
13 $\quad$ **si** $\omega \neq \emptyset$ **alors** $\Omega_3.add(\omega)$
fin
14 $Loc.addAll(\Omega_1)$; $Loc.add(\Omega_2)$; $Loc.addAll(\Omega_3)$;
retourner Loc ;

5 Expérimentations

Cette section introduit les différentes études expérimentales que nous avons réalisées.

Nous utilisons dans nos expérimentations la suite de programmes de Siemens, qui est la base la plus utilisée pour la comparaison des différentes techniques

TABLE 1 – La base de programmes Siemens

Programme	Versions erronées	LOC [2]	$\|\mathcal{T}^+\|$	$\|\mathcal{T}^-\|$
PrintTokens	4	472	4016	55
PrintTokens2	9	399	3827	229
Replace	29	512	5450	92
Schedule	5	292	2506	144
Schedule2	8	301	2646	34
Tcas	37	141	1015	37
TotInfo	19	440	1542	36

de localisation de fautes. Cette base est composée de sept programmes écrits en C, où chaque programme se décline en plusieurs versions ayant différentes fautes. Notons que des suites de cas de test sont disponibles pour tester chaque catégorie de programmes [1]. Lors de nos expérimentations, nous avons exclu 21 versions qui sont hors de portée de la tâche de localisation de fautes (p. ex., erreurs de segmentation). En somme, nous avons 111 programmes avec des fautes, détaillés dans la table 1, où nous donnons pour chaque programme le nombre de versions erronées, le nombre de lignes de code et le nombre moyen de cas de test positifs $|\mathcal{T}^+|$ et négatifs $|\mathcal{T}^-|$.

Tout d'abord, nous avons besoin de savoir quelle instruction est couverte par une exécution donnée. Pour cela, nous faisons appel à l'outil GCOV [3]. GCOV peut être aussi utilisé pour connaître le nombre d'exécutions d'une instruction. Ceci dit, dans notre approche nous avons seulement besoin de la matrice de couverture dont la génération nécessite l'exécution du programme P sur chaque cas de test, puis le résultat retourné est comparé avec le résultat attendu. Si les deux résultats sont égaux, nous ajoutons la couverture du cas de test à la base transactionnelle positive, autrement la couverture est ajoutée à la base négative.

Notre mise en œuvre informatique est nommée F-CPMINER. L'implémentation a été réalisée en utilisant GECODE [4], un solveur ouvert, libre et portable, pour le développement des programmes à contraintes. Notre implémentation inclut l'algorithme 1 pour l'extraction des top-k motifs suspects (voir Sect. 4) et l'algorithme 2 qui traitent les top-k motifs et retourne une localisation précise de fautes avec un classement des instructions selon leurs degrés de suspicion. Nos expérimentations ont été réalisées avec un processeur Intel Core i5-2400 3.10 GHz et 8 GO de RAM sous Ubuntu 12.04 LTS.

Nous avons implémenté la méthode TARANTULA et avons évalué la suspicion des instructions de la même manière que celle présentée dans [9]. En particulier, nous avons adopté une métrique très connue dans la localisation de fautes, à savoir l'EXAM score [20] qui mesure l'efficacité d'une approche de localisation donnée. L'EXAM score donne le pourcentage des instructions qu'un développeur doit vérifier avant d'atteindre celle contenant la faute. Il est donc logique que la meilleure méthode est celle qui a le pourcentage d'EXAM score le plus petit.

1. une description de Siemens suite est donnée dans [8]
2. nombre de lignes de code
3. `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`
4. `www.gecode.org`

TABLE 2 – F-CPMINER versus TARANTULA

		P-EXAM		O-EXAM		Δ-EXAM	
Programme	k	F-CPMINER	TARANTULA	F-CPMINER	TARANTULA	F-CPMINER	TARANTULA
PrintTokens	195	**5.89**	13.58	**3.97**	11.66	1.92	1.92
PrintTokens2	200	**2.66**	16.44	**1.72**	15.50	0.94	0.94
Replace	245	12.34	**11.04**	10.47	**9.31**	1.87	**1.73**
Schedule	152	25.26	**6.71**	22.63	**5**	2.63	**1.71**
Schedule2	128	**44.82**	62.01	**30.46**	51.17	14.36	**10.84**
TotInfo	123	**11.51**	23.62	**5.56**	17.88	5.95	**5.74**
Tcas	65	**40.33**	43.95	**16.13**	20.12	24.19	**23.83**

TARANTULA et F-CPMINER peuvent retourner un ensemble d'instructions équivalentes en terme de suspicion (c.-à-d., avec le même degré de suspicion). Dans ce cas, l'efficacité dépend de l'instruction qui est examinée en premier. Pour cette raison, nous donnons deux EXAM score, l'EXAM score *optimiste* et l'EXAM score *pessimiste*, notés respectivement O-EXAM et P-EXAM. Nous parlons de O-EXAM (resp. P-EXAM) lorsque la première (resp. la dernière) instruction à examiner dans l'ensemble des instructions équivalentes, est l'instruction contenant la faute. Nous définissons également une troisième métrique, le Δ-EXAM = O-EXAM- P-EXAM, représentant la marge de l'EXAM score. En d'autres termes, Δ-EXAM représente la distance entre le score optimiste et le score pessimiste. Un point important de l'expérimentation, est le choix de la valeur de k. Dans [14] une expérimentation montre clairement que lorsque la valeur de k est fixée au nombre d'instructions du programme, on obtient un Δ-EXAM très réduit, ce qui signifie que les ensembles d'instructions équivalentes en terme de suspicion sont réduits, ce qui conduit à une localisation plus précise.

5.1 F-CPMINER versus TARANTULA

La table 2 donne une comparaison entre F-CPMINER et TARANTULA basée sur l'EXAM score sur les 111 programmes de la suite Siemens. Pour chaque classe de programmes (p. ex., Tcas-* inclut 37 versions), nous rapportons la valeur de k utilisée dans F-CPMINER pour extraire les top-k motifs suspects (correspondant à la taille du programme), les valeurs moyennes de P-EXAM, O-EXAM et Δ-EXAM.

La première observation est que F-CPMINER est meilleur dans 5 classes de programmes sur 7 selon les valeurs de P-EXAM et O-EXAM rapportées. Par

exemple, si nous prenons la classe PrintTokens2-*, avec F-CPMINER la faute est localisée après avoir examiné seulement 2.66% du code dans le cas pessimiste (P-EXAM) et seulement 1.72% dans le cas optimiste (O-EXAM). Cependant, TARANTULA a besoin d'examiner 16.44% dans le cas pessimiste et 15.5% du code dans le cas optimiste.

Notre seconde observation concerne les valeurs de Δ-EXAM qui sont en général meilleures dans TARANTULA que F-CPMINER. Toutefois, cette différence n'excède pas 4% pour une classe de programmes et 1% pour les autres classes par rapport à TARANTULA.

Dans le but de compléter les résultats donnés par la table 2, nous rapportons dans la table 3 une comparaison par paires entre F-CPMINER et TARANTULA sur les 111 programmes. Pour chaque catégorie de programmes (p. ex., Tcas), nous donnons le nombre de programmes où F-CPMINER ou TARANTULA est meilleur en comparant les valeurs des EXAM score retournés. Nous donnons aussi le cas où les valeurs de l'EXAM score sont égaux pour les deux méthodes (ÉQUIVALENTES).

Les résultats de la table 3 correspondent avec ceux de la table 2, où F-CPMINER a une meilleur efficacité dans 5 classes de programmes sur 7. Par exemple, pour la classe de programmes Tcas et en comparant les valeurs de O-EXAM (resp. P-EXAM), F-CPMINER est plus précis dans 19 (resp. 20) programmes avec fautes, alors que TARANTULA est meilleur dans seulement 6 (resp. 4) programmes. Enfin, les deux approches sont équivalentes en obtenant le même O-EXAM (resp. P-EXAM) dans 12 (resp. 13) programmes.

A partir des tables 2 et 3, l'observation générale est que F-CPMINER est très concurrentiel par rapport à TARANTULA en terme d'efficacité (soit en pessimiste ou en optimiste). Cela est particulièrement vrai où F-CPMINER gagne au total sur 60% des programmes considérés dans le benchmark (61 à 62 sur 111 programmes), là où TARANTULA fait mieux sur seulement 25% (24 à 29 sur 111 programmes). Enfin, les deux approches se comportent de manière similaire sur 15% des programmes (21 à 25 sur 111 programmes).

La figure 2 montre une comparaison de l'efficacité basée sur le P-EXAM et O-EXAM entre F-CPMINER et TARANTULA. L'axe des x donne les valeurs de l'EXAM score et l'axe des y donne le pourcentage cumulatif des fautes localisées sur les 111 programmes. Pour le cas pessimiste (c.-à-d., P-EXAM). Jusqu'à une valeur de 10%, les deux outils agissent de la même manière en localisant plus de 40% des fautes. De 10% à 30%, nous observons une différence de plus de 10% de fautes localisées en faveur de F-CPMINER comparé à TARANTULA (60% pour F-CPMINER au lieu de 50% des fautes localisées pour TARANTULA). Cette différence est réduite à 6% dans l'intervalle de 30% à 40% de l'EXAM score. Entre 60% et 70% de l'EXAM score, nous observons que TARANTULA a rattrapé F-CPMINER en localisant plus de 84% des fautes. Toutefois, les 20% des fautes restantes sont rapidement localisées par F-CPMINER (de 70% à 80% de l'EXAM score) alors que TARANTULA continue jusqu'à la fin pour localiser toutes les fautes (jusqu'à 100% de l'EXAM score). Pour le cas optimiste (c.-à-d., O-EXAM) illustré par les courbes en pointillées, F-CPMINER agit assez rapidement dès le début en localisant plus de fautes que TARANTULA. Il est aussi important de noter que les deux courbes ne s'intersectent pas et que celle de F-CPMINER est toujours au dessus de celle de TARANTULA. Notons aussi que après 30% de l'EXAM score, F-CPMINER dans le cas pessimiste localise le même pourcentage de fautes (c.-à-d., 74%) que TARANTULA dans le cas optimiste.

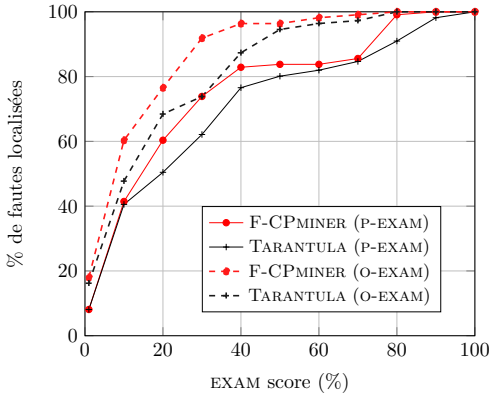


FIGURE 2 – TARANTULA et F-CPMINER : comparaison d'efficacité

Dans le but de renforcer les résultats précédents, nous avons mené le test d'hypothèse des *rangs signés de Wilcoxon*, ce choix est dû au fait que nous ne sommes pas en mesure d'assumer une distribution normale de la population [17]. Ici, l'hypothèse alternative unilatérale est considérée avec l'hypothèse nulle suivante (TARANTULA est plus efficace que F-CPMINER) H_0 : l'EXAM score retourné par F-CPMINER $\geq$ l'EXAM score retourné par TARANTULA. L'hypothèse alternative H_1 déclare que F-CPMINER examine un nombre plus petit d'instructions que TARANTULA pour localiser les fautes (F-CPMINER est plus efficace que TARANTULA). Le test mené, nous a permis de conclure que l'hypothèse H_1 est acceptée avec un taux de confidence égale à $79,99\%$ dans le cas pessimiste (P-EXAM), et un taux de $97,99\%$ dans le cas optimiste (O-EXAM). Ce test confirme les résultats précédents de la comparaison de F-CPMINER avec TARANTULA.

D'une manière générale, nous pouvons conclure que F-CPMINER est capable de localiser la plupart des

TABLE 3 – Comparaison par paires entre F-CPMINER et TARANTULA.

	Print	Print2	Replace	Sched	Sched2	Tcas	Totinfo	**Total**
F-CPMINER	**(2,2)**	**(6,6)**	(12,12)	(1,1)	**(7,7)**	**(19,20)**	**(14,14)**	**(61,62)**
TARANTULA	(0,0)	(0,0)	**(16,15)**	**(4,4)**	(1,1)	(6,4)	(2,0)	(29,24)
ÉQUIVALENTS	(2,2)	(3,3)	(1,2)	(0,0)	(0,0)	(12,13)	(3,5)	(21,25)
								(P-EXAM, O-EXAM)

fautes plus rapidement que TARANTULA en terme d'efficacité d'EXAM score.

Un autre résultat intéressant, en considérant l'EXAM score cumulatif sur les 111 programmes, nous donnons le nombre total d'instructions qui doivent être examinées par les deux méthodes sur l'ensemble des 111 programmes. F-CPMINER doit examiner 2861 instructions sur un total de 16211 instructions contre 3482 pour TARANTULA dans le cas du P-EXAM (un gain de 17%). Dans le cas du O-EXAM, F-CPMINER doit examiner 1808 instructions contre 2496 pour TARANTULA (un gain de 27%).

Enfin, nous concluons cette section avec les observations suivantes sur les 111 programmes testés :

- Dans 110 programmes, l'instruction fautive est dans le premier motif (c.-à-d., le plus suspect $\mathcal{SM}_1$).
- Dans 96 programmes, la faute est une instruction qui disparaît de $\mathcal{SM}_1$ qui est donc dans Ω_1.
- Dans 14 programmes, la faute est localisée dans Ω_2.
- Dans seulement un seul programme, la faute est localisée dans Ω_3.

Ces observations montrent clairement l'efficacité de la stratégie de classement adoptée dans l'algorithme 2.

5.2 Analyse du temps CPU

Dans cette section nous faisons une analyse du temps CPU de notre approche. Il est important de rappeler que l'approche TARANTULA évalue le degré de suspicion de chaque instruction en utilisant une formule probabiliste sans prendre en compte aucune dépendance entre instructions. Ainsi, l'explosion combinatoire due aux combinaisons possibles d'instructions n'est pas traitée par TARANTULA. Par conséquent, le temps CPU obtenu par TARANTULA est négligeable (en millisecondes).

La table 4 rapporte pour chaque classe de programme le temps CPU moyen et l'écart type pour les deux étapes de F-CPMINER (c.-à-d., l'extraction des top-k motifs correspond à l'algorithme 1 et l'étape de post-traitement correspond à l'algorithme 2).

La première observation que l'on peut faire est que l'extraction des top-k motifs est l'étape la plus coûteuse dans F-CPMINER. Ceci étant en partie expliqué par le très grand nombre de motifs candidats (c.-à-d., $|\mathcal{L}_\mathcal{I}|$). Toutefois, dans nos expérimentations, les temps CPU ne dépassent pas 235 secondes dans le pire des cas (programme Replace) et moins de 0.2 secondes dans le meilleur des cas (programme Tcas). La seconde observation est que le temps CPU consommé par l'étape de post-traitement pour la localisation de l'instruction fautive est négligeable (de l'ordre de millisecondes dans le meilleur et le pire des cas).

TABLE 4 – Comparaison des temps CPU pour les deux étapes de F-CPMINER sur la suite Siemens (en secondes).

Programme	top-k extraction	top-k traitement
PrintToken	**61.49** ± 34.35	**0.030** ± 0.003
PrintToken2	**146.25** ± 65.89	**0.045** ± 0.013
Replace	**147.69** ± 86.02	**0.051** ± 0.015
Schedule	**27.41** ± 15.28	**0.012** ± 0.004
Schedule2	**12.66** ± 05.37	**0.016** ± 0.004
TotInfo	**2.53** ± 01.01	**0.014** ± 0.006
Tcas	**0.16** ± 00.02	**0.001** ± 0.000

6 Conclusion

Dans ce papier nous avons présenté une nouvelle approche basée sur la fouille de motifs ensemblistes et la programmation par contraintes (PPC) pour traiter le problème de la localisation de fautes. Dans un premier temps, nous avons formellement défini le problème de la localisation de fautes comme une problématique de fouille de données. Dans un second temps, nous avons proposé une approche qui procède en deux étapes. La première étape fait appel à la PPC pour modéliser et résoudre les contraintes tenant compte de la nature bien particulière des motifs recherchés. La résolution du modèle à contraintes nous permet d'obtenir les top-k suites d'instructions les plus suspectes. La seconde étape ad-hoc a pour objectif de raffiner la localisation en classant de façon plus précise l'ensemble des instructions suspectes. Finalement, nous avons détaillé une étude expérimentale qui compare notre outil F-CPMINER, une implémentation de l'approche proposée, avec l'approche standard TARANTULA sur la base de programmes Siemens. Les résultats montrent que notre approche permet de proposer une localisation plus précise.

Comme travaux futurs, nous prévoyons d'explorer d'autres observations sur le comportement d'un programme avec faute et enrichir le modèle avec plus de contraintes pour la localisation.

Références

[1] Peggy Cellier, Mireille Ducassé, Sébastien Ferré, and Olivier Ridoux. Dellis : A data mining process for fault localization. In *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009), Boston, Massachusetts, USA, July 1-3, 2009*, pages 432–437, 2009.

[2] Peggy Cellier, Mireille Ducassé, Sébastien Ferré, and Olivier Ridoux. Multiple fault localization with data mining. In *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2011), Eden Roc Renaissance, Miami Beach, USA, July 7-9, 2011*, pages 238–243, 2011.

[3] B. Crémilleux and A. Soulet. Discovering knowledge from local patterns with global constraints. In *ICCSA (2)*, pages 1242–1257, 2008.

[4] Luc De Raedt, Tias Guns, and Siegfried Nijssen. Constraint programming for itemset mining. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 204–212. ACM, 2008.

[5] Luc. De Raedt and A. Zimmermann. Constraint-based pattern set mining. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, Minneapolis, Minnesota, USA, April 2007. SIAM.

[6] W Eric Wong, Vidroha Debroy, and Byoungju Choi. A family of code coverage-based heuristics for effective fault localization. *Journal of Systems and Software*, 83(2) :188–208, 2010.

[7] Tias Guns, Siegfried Nijssen, and Luc De Raedt. Itemset mining : A constraint programming perspective. *Artificial Intelligence*, 175(12) :1951–1983, 2011.

[8] Monica Hutchins, Herb Foster, Tarak Goradia, and Thomas Ostrand. Experiments of the effectiveness of dataflow-and controlflow-based test adequacy criteria. In *Proceedings of the 16th international conference on Software engineering*, pages 191–200. IEEE Computer Society Press, 1994.

[9] James A. Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), November 7-11, 2005, Long Beach, CA, USA*, pages 273–282, 2005.

[10] James A. Jones, Mary Jean Harrold, and John T. Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 22rd International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida, USA*, pages 467–477, 2002.

[11] M. Khiari, P. Boizumault, and B. Crémilleux. Constraint programming for mining n-ary patterns. In *CP'10*, volume 6308 of *LNCS*, pages 552–567. Springer, 2010.

[12] J.C. C. Laprie, A. Avizienis, and H. Kopetz, editors. *Dependability : Basic Concepts and Terminology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.

[13] Mehdi Maamar, Nadjib Lazaar, Samir Loudni, and Yahia Lebbah. Localisation de fautes à l'aide de la fouille de données sous contraintes. Colloque sur l'Optimisation et les Systèmes d'Information COSI, 2015.

[14] Mehdi Maamar, Nadjib Lazaar, Samir Loudni, and Yahia Lebbah. Fault localization using itemset mining under constraints. *Automated Software Engineering*, page Forthcoming, 2016.

[15] Syeda Nessa, Muhammad Abedin, W Eric Wong, Latifur Khan, and Yu Qi. Software fault localization using n-gram analysis. In *Wireless Algorithms, Systems, and Applications*, pages 548–559. Springer, 2008.

[16] P. Kralj Novak, N. Lavrac, and G. I. Webb. Supervised descriptive rule discovery : A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research*, 10 :377–403, 2009.

[17] R Ott and Micheal Longnecker. *An introduction to statistical methods and data analysis*. Cengage Learning, 2008.

[18] Manos Renieres and Steven P Reiss. Fault localization with nearest neighbor queries. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, pages 30–39. IEEE, 2003.

[19] Willy Ugarte Rojas, Patrice Boizumault, Samir Loudni, Bruno Crémilleux, and Alban Lepailleur. Mining (soft-) skypatterns using dynamic CSP. In *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings*, volume 8451 of *Lecture Notes in Computer Science*, pages 71–87. Springer, 2014.

[20] W Eric Wong and Vidroha Debroy. A survey of software fault localization. *Department of Computer Science, University of Texas at Dallas, Tech. Rep. UTDCS-45-09*, 2009.

Aberrations dans les Problèmes SAT

Gilles Audemard[1] **Laurent Simon**[2]

[1] Univ. Lille-Nord de France – CRIL/CNRS UMR8188 – Lens, F-62307, France
[2] Université de Bordeaux, LABRI
audemard@cril.fr lsimon@labri.fr

Résumé

Avec l'amélioration continue des démonstrateurs SAT, de nombreux problèmes issus de domaines très différents ont été ajouté à la catégorie des problèmes applicatifs, l'un des jeux de tests de référence sur lequel sont généralement comparés les solveurs. Ces problèmes sont tout aussi importants lorsque l'on désire mesurer l'impact (supposé positif) d'une nouvelle idée. Nous montrons dans cet article que, parmi ces problèmes, nombreux sont ceux qui exhibent des comportements extrêmes, des aberrations. Partant de quelques indicateurs très simples, nous montrons comment laisser notre solveur `Glucose` choisir parmi quatre stratégies afin de résoudre efficacement ces problèmes extrêmes. Ensuite, nous utilisons une nouvelle stratégie de choix de polarité qui améliore encore les résultats. Plus intéressant, sans cette spécialisation, cette nouvelle heuristique s'avère inefficace montrant l'impact néfaste que peuvent avoir les cas extrêmes sur le comportement d'un solveur.

Abstract

With the increasing performance of SAT solvers, a lot of distinct problems, coming from very disparate fields, are added to the pool of *Application* problems, regularly used to rank solvers. These problems are widely used to measure the positive impact of any new idea. We show in this paper that a number of those problems have extreme behaviors that any SAT solvers must deal with. We show that, by adding a few, simple, human-readable indicators, we can let `Glucose` choose between four strategies to obtain important improvements on the set of the hardest problems from all the competitions between 2002 and 2013 included. Moreover, once the SAT solver has been specialized, we show that a new restart polarity policy can improve even more the results. Without the specialization step mentioned above, this new and effective policy would have been jugged inefficient. Our final `Glucose` is capable of solving 20% more problems than the original one, while speeding up also UNSAT answers.

1 Introduction

À la fin des années 90, les problèmes (*benchmarks*) utilisés pour évaluer les démonstrateurs SAT provenaient du même ensemble de problèmes. Quelques années plus tard, avec l'introduction des solveurs CDCL [16, 17, 9], est apparu la nécessité de les partitionner en problèmes *aléatoires*, "*crafted*" (conçus pour être typiquement difficile à résoudre) et "*industriels/applications*". Il était déjà évident qu'une technique développée pour une catégorie de problèmes donnée ne serait pas efficace sur une autre [8, 11, 3]. Ainsi, la première compétition SAT (2002) a utilisée cette partition pour récompenser les démonstrateurs. Cette partition, d'un haut niveau, est toujours d'actualité.

Néanmoins, avec l'intérêt croissant pour la technologie SAT, de nombreux problèmes provenant de domaines aussi divers que la cryptographie, la biologie, les graphes, ont été réduits à SAT. Tous ces problèmes, ni *aléatoires*, ni *crafted* sont inclus dans la catégorie *application* [5], qui contient dès lors des problèmes très disparates. Cela ne serait pas problématique si la recherche autour de SAT n'était pas autant axée sur les résultats empiriques : toute nouvelle idée doit être validée expérimentalement avec les benchmarks de la compétition précédente, qui utilise ces propres règles pour les sélectionner. Ainsi, une nouvelle idée peut avoir impact important une année donnée et se révéler totalement inintéressante l'année suivante, suivant l'ensemble de problèmes sélectionnés.

Dans cet article, nous montrons que cette ancienne classification des problèmes peut s'avérer contre-productive. Elle est basée sur notre expérience autour de `Glucose` [4, 2, 3] et des derniers développements autour des démonstrateurs SAT [6, 18, 7, 22]. Il nous est très difficile d'améliorer `Glucose` sur l'ensemble des problèmes. Dans la plupart des cas, ce qui est gagné d'un coté est perdu de l'autre.

Les contributions de cet article s'articulent autour de deux axes. Tout d'abord, nous montrons que de nombreuses séries de problèmes exhibent des comportements extrêmes, des aberrations. Nous avons lancé `Glucose` sur de nombreux benchmarks et avons collecté certaines informations caractérisant la recherche (nombre de décisions entre chaque conflit, profondeur moyenne à laquelle un conflit survient...). Nous montrons que de nombreuses familles de benchmarks ont des comportements atypiques. En nous focalisant sur ces comportements extrêmes, nous avons proposé des heuristiques adaptatives à notre solveur `Glucose`. Ensuite, nous nous sommes focalisé sur la phase (la polarité de choix des variables de décision) [20] à utiliser lors des redémarrages. Cette nouvelle heuristique est motivée par de nombreuses expérimentations montrant que la phase peut être découpée en deux forces (une pour SAT et une pour UNSAT) ouvrant à une nouvelle vision des démonstrateurs CDCL. Ce qui est surprenant dans nos résultats, c'est que ce mécanisme est sans intérêt pour la version classique de `Glucose`, mais s'avère très efficace avec `Glucose` adaptatif.

L'article est construit ainsi : dans la section 2, nous montrons que de nombreux problèmes exhibent des comportements extrêmes. Nous introduisons ensuite une politique adaptative à `Glucose` pour essayer de passer outre ces comportements atypiques. Dans la section 3, nous présentons une nouvelle heuristique de choix de polarité. La section 4 présente des résultats empiriques montrant que la classification actuelle des benchmarks peut être trompeuse. Avant de conclure, nous débattons dans la section 5 de nos résultats et des dernières orientations autour de la résolution SAT.

2 Problèmes SAT : gérer des cas extrêmes

Comme nous allons le voir dans cette section, l'ensemble des benchmarks considéré dans les compétitions SAT est constitué de nombreuses aberrations, *i.e.*, des problèmes aux comportements extrêmes. Ces problèmes spécifiques ne sont pas des exceptions par eux-mêmes, mais font partie de familles d'exceptions. Il a déjà été mentionné que les problèmes de la catégorie *application* peuvent être séparé en différentes catégories [5]. Néanmoins, cette partition a été proposée en prenant uniquement en compte l'origine des problèmes (cryptographie, vérification de circuits...) et non pas le comportement des solveurs sur ceux-ci. Cette série de benchmarks atypiques peut s'avérer très problématique : une simple série contenant de nombreux benchmarks peut complètement changer le résultat d'une compétition, ou, de manière plus embêtante, donner une fausse impression sur une nouvelle technique de résolution. Nous reviendrons sur cela dans la section 5.

Afin de détecter correctement ces aberrations, nous avons récupéré l'ensemble des problèmes utilisés dans les catégories *industrial/application* (des compétitions allant de 2002 à 2013) et avons fait de notre mieux pour supprimer les doublons. Nous obtenons ainsi un ensemble de 2632 problèmes. Dans la première partie de l'article nous n'avons considéré que les problèmes qui n'ont pas été résolu en 100000 conflits après prétraitement (nous utilisons Satelite comme outil de prétraitement) et nous avons alors collecté différentes statistiques. Cela donne un ensemble de 1164 problèmes difficiles. Cette section est donc basé sur l'étude de ces 1164 problèmes [1].

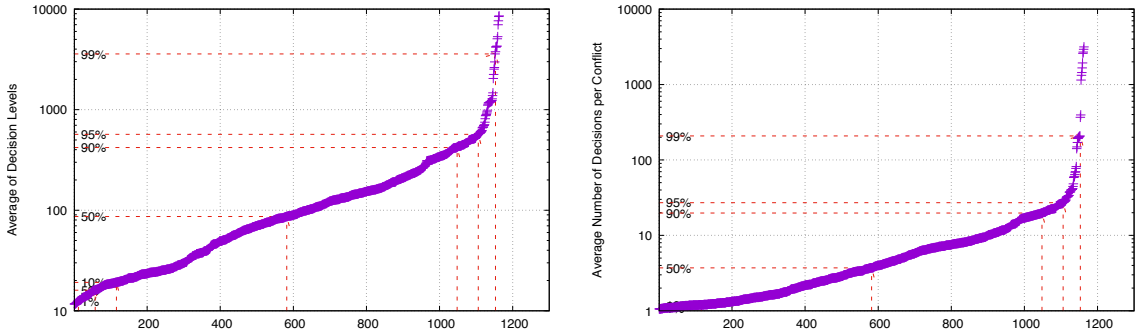
2.1 Des aberrations, toujours des aberrations

Nous avons donc collecté des statistiques, considérées comme mesures témoins du comportement d'un démonstrateur SAT. Ces mesures sont le nombre de décisions, le nombre de décisions entre deux conflits, le nombre de conflits successifs, le nombre de clauses glues non binaires et le nombre de propagations.

2.1.1 Niveau de décision

Le premier indicateur que nous proposons d'observer est le niveau de décisions où les conflits surviennent. Nous avons donc fait une moyenne (pour chaque problème) sur les 100000 premiers conflits et reportons la distribution de cette mesure sur la Figure 1(a). 10% des problèmes nécessitent plus de 400 décisions (en moyenne) pour atteindre un conflit, et 10% moins de 30 décisions. Il est clair qu'un démonstrateur SAT ne peut pas avoir les même comportement sur ces problèmes extrêmes. On peut raisonnablement supposer qu'une technique qui va marcher sur le premier type de benchmarks ne marchera plus sur le second (et vice-versa). Nous pouvons, par exemple, nous questionner sur le retour non-chronologique et l'apprentissage sur des problèmes nécessitant très peu de décisions avant d'atteindre un conflit. Plus intéressant encore, de nombreux benchmarks atypiques font, en fait, parti de familles de problèmes : les problèmes avec les plus grands niveaux de décisions font partis de la famille hwmcc12miters (compétition 2013), li-test4 (2003), 9dlx-vliw (2004). A l'opposé, avec les niveaux de décisions le plus faibles, on trouve les familles longmult (2008), desgen-gss (2009), kummling-grosmann-ctl (2013), traffic (2011), eq.atree.braun (2007). Il est important de noter que certains des problèmes difficiles peuvent exhiber un niveau moyen de décisions

1. La liste des problèmes ainsi que des informations additionnelles sont disponibles à `http://www.labri.fr/perso/lsimon/sat16`



(a) Niveau de décision moyen où un conflit survient

(b) Nombre moyen de décisions entre chaque conflits

FIGURE 1 – Distribution cumulative auprès des 1162 problèmes. Statistiques collectées après 100000 conflits. Notez l'échelle logarithmique sur l'axe des ordonnées.

plus petit que 20, montrant une structure très *combinatoire*.

2.1.2 Nombre de décisions par conflits

Une autre mesure très simple consiste à comptabiliser le nombre de décisions effectuées lors des 100000 premiers conflits. Cette mesure nous donne une approximation du nombre de décisions réalisées entre deux conflits successifs (si nous faisons abstraction des redémarrages). Une grande valeur identifie des problèmes où les conflits sont éloignés les uns des autres.

La Figure 1(b) montre la distribution de cette mesure. Il est frappant de noter que pour la moitié des benchmarks seules 3 décisions sont réalisées entre deux conflits successifs. A l'opposé 10% des problèmes nécessitent plus de 20 points de choix, ce qui est étonnamment important. Clairement, cela montre une large hétérogénéité entre les problèmes. Ici encore, ce sont des familles d'instances qui ont des comportements extrêmes. Les instances des familles bivium (2013), hitag (2013), par32 (2002), desgen-gss (2009) ont moins de 1.1 décisions par conflits en moyenne. Ces familles exhibent clairement une structure *combinatoire* ce qui est cohérent avec leur nature. Dans ces problèmes, l'analyse de conflit ne peut donner des informations importantes sur le retour arrière. A l'opposé, nous trouvons hwmcc12miters (2013), li-test4 (2003), 9dlx-vliw (2004) qui nécessitent entre 30 et 3 000 décisions par conflits, ce que nous ne pensions pas possible.

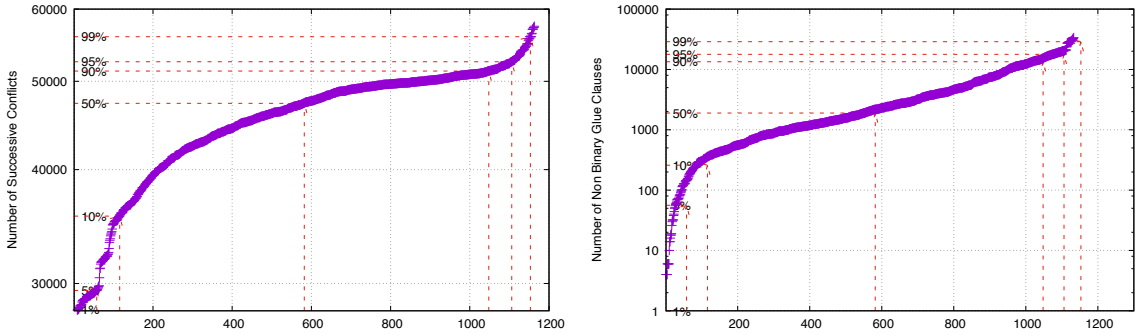
2.1.3 Conflits successifs

La Figure 2(a) montre le nombre de conflits successifs, c.a.d. les conflits qui surviennent directement après un conflit, sans décision intermédiaire. Un nombre important de conflits successifs peut mettre en avant des problèmes pour lesquels la résolution et le FUIP ne sont pas assez puissants pour analyser les conflits. Cette figure exhibe un comportement moins pathologique que les deux précédentes. Malgré tout, le premier et le dernier décile montrent clairement des comportements extrêmes. Les faibles valeurs apparaissent sur la famille nossum-sha1 (2013) avec environ 30 000 conflits successifs (sur 100000 conflits), suivie par la famille (très difficile à résoudre) MD5 (2014). A l'opposé, le nombre de conflits successifs le plus important est observé sur les vmpc (2006) et les bio-rpcl-xits (2009). Il est important de noter que dans tous les cas, au moins 30% des conflits surviennent sans aucune décision ce qui est vraiment important [2].

2.1.4 Clauses glues non binaires

Les clauses glues constituent l'élément principal de `Glucose`. Sur certains problèmes, les solveurs apprennent de nombreuses clauses binaires (qui sont triviallement glues). Mesurer le nombre de clauses glues qui ne sont pas binaires peut nous donner des indices supplémentaires sur la nature d'un problème. De plus, la stratégie de `Glucose` oblige le solveur à garder indéfiniment toutes les clauses glues, alors que, habituellement, les démonstrateurs ne conservent indéfiniment que les clauses binaires. On peut donc espérer que les problèmes générant des clauses glues non binaires soient plus faciles pour `Glucose` que pour d'autres solveurs. La Figure 2(b) montre la distribution cumulative de cette mesure. Les familles Homer (2002), stable (2014), ndhf-xits (2009) n'apprennent aucune clause glue durant les 100000 premiers conflits. La famille bivium (2013) en apprend moins de 128 et la famille hitag (2013) moins de 300 (ces familles

2. Cette observation a été initialement faite par Lakhdar Saïs (CRIL, Lens) durant l'une des réunions de l'ANR UNLOC



(a) Conflits successifs (sans décision)

(b) Nombre de clauses glues non binaires

FIGURE 2 – Distribution cumulative auprès des 1162 problèmes. Statistiques collectées après 100000 conflits. Notez l'échelle logarithmique sur l'axe des orsdonnées.

de problèmes s'avèrent très difficiles à résoudre par `Glucose`). A l'opposé, les benchmarks ibm-SAT-datk (2002-2012) apprennent plus de 20 000 clauses glues (non binaires) et les goldberg-bmc2-cnt10 (2002) plus de 30 000. On retrouve ici les typiquement des problèmes de Bounded Model Checking.

2.1.5 Propagations unitaires

Le nombre de littéraux unitaires propagés durant la recherche est une mesure importante pour évaluer les performances d'un solveur SAT. Nous montrons ici que, suivant les problèmes, il existe des grandes différences dans cette mesure. La Figure 3(a) reporte la distribution sur le nombre de propagations réalisées lors des 100000 premiers conflits. Ici aussi, certaines familles exhibent des comportements extrêmes. Les familles Homer (2002), stable (2014), aes (2011) effectuent le moins de propagations. Les plus grandes valeurs surviennent avec les familles atco (2014), arcfour (2013), hwmcc12miters (2013), ibm-sat-dat, ACG (2009) et gss (2009). Dans les cas les plus extrêmes, on peut propager jusqu'à 10 000 littéraux avant entre chaque conflit. Pour résoudre plus efficacement ces problèmes, il est peut être nécessaire de réaliser un pré-traitement plus efficace, de rechercher les équivalences entre littéraux, ou les littéraux impliqués [13].

2.2 Autres mesures

Nous détaillons dans la table 3(b) une sélection d'autres mesures que nous avons réalisées, afin de mettre en avant que sur chacune d'entre elles des comportements aberrants peuvent être observés. La colonne BJ donne des résultats sur le niveau de backjump (le nombre de décisions supprimées lors du saut arrière), DL le niveau de décision, LBD la valeur moyenne des LBD des clauses apprises, Size leur taille, BR le nombre de redémarrages bloquées et FUIPisDEC le pourcentage de cas où le FUIP est également la variable de décision. Les données de cette table montrent des valeurs de kurtosis et de skwness importantes, ce qui est typiquement le témoin de comportements extrêmes.

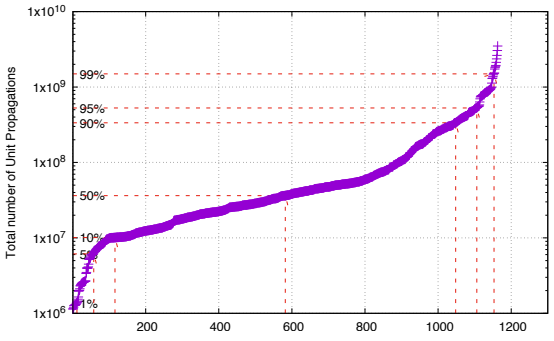
2.3 Synthèse : déterminer la meilleure stratégie

Nous venons donc de mettre en évidence que les démonstrateurs CDCL (ici `Glucose`) ont des comportements très disparates sur les problèmes de la catégorie *application*. Dès lors, il n'est pas réaliste, d'espérer avoir une approche CDCL générale qui soit efficace sur tous ces problèmes. Nous avons alors décider d'adapter le comportement de `Glucose` sur ces problèmes extrêmes, de manière indépendante. Une fois identifier les benchmarks aberrants, nous avons modifié les paramètres du solveur sur chaque comportements extrêmes. Nous donnons par la suite, les configurations que nous avons retenues.

2.3.1 Peu de décisions

Pour ce type de problème, la stratégie utilisée par Chanseok Oh [18] s'avère l'une des plus efficace. Dans ce cas, nous conservons toutes les clauses qui ont un LBD plus petit que 4 et utilisons sa stratégié de réduction de la base de clauses apprises, i.e. nous utilisons l'activité des clauses telle que définie dans la version originale de `Minisat`. Avec cette simple modification, `Glucose` s'avère capable de résoudre 20 (sur 30) problèmes Bivium/Hitag (en lieu et place de 1 pour la version classique de `Glucose`!).

Il y a toutefois des différences avec la technique proposée dans [18]. Nous n'alternons pas entre deux stra-



(a) Nombre de littéraux unitaires propagés

Measure	BJ	DL	LBD	Size	BR	FUIPisDec
Min	1	10	4	5	0	24%
Max	3155	8559	84	3395	1730	91%
Median	3	87	10	25	298	61%
Mean	24	218	14	74	34	63%
St. Dev.	195	623	11	242	340	12%
Skewness	13	9	3	9	1	0.3 %
Kurtosis	174	99	14	100	6	2 %

(b) Autres statistiques

FIGURE 3 – Différentes mesures collectées après 100000 conflits sur 1162 problèmes.

tégies. Si elle est choisie (après 100000 conflits), cette stratégie sera utilisée jusqu'à la fin de la recherche. Cela est d'ailleurs le cas pour toutes les autres adaptations que nous proposons.

2.3.2 Conflits successifs

Si ce nombre est faible (comme cela est le cas pour les instances nossum et MD5), comme souligné dans [18] nous utilisons un redémarrage de type Luby [14] et une constante VSDIDS très grande (0.999 au lieu de 0.95). Nous restons dans l'incapacité de résoudre des instances MD5 mais somme capable de résoudre 25 instances nossum de plus.

Si cette valeur est très grande, nous avons testé différentes valeurs pour essayer de résoudre les instances VMPC. Ces problèmes de cryptographie sont assez difficiles. C'est en utilisant la stratégie proposée dans `minisat-blbd`, qui diversifie la recherche entre chaque redémarrages, que nous avons obtenu les meilleurs résultats.

2.3.3 Clauses glues non binaires

Dans ce cas, nous avons observé que l'on devait diminuer la valeur de la constante VSIDS (0.91), ce qui a pour effet d'oublier beaucoup plus rapidement les conflits antérieurs, pour obtenir de bons résultats sur les instances ibm. Les raisons de ce choix nous sont encore floues.

3 Modifier la phase

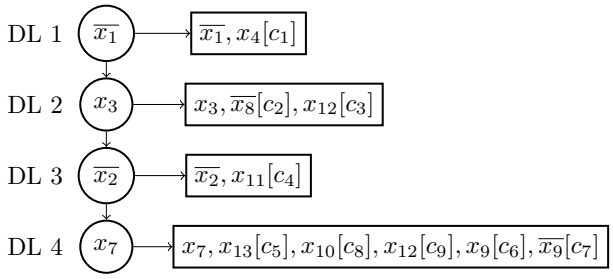
Le mécanisme de *phase saving* est un composant essentiel des démonstrateurs SAT [20], tout particulièrement lorsque les redémarrages sont très fréquents. Malgré tout, il y a toujours des questions autour de son efficacité. Par exemple, *La phase favorise -t-elle l'obtention d'un modèle (SAT) ou la contradiction (UNSAT) ?*. En effet, la phase est mise à jour au moment de l'analyse de conflit (ce qui suggère que l'on cherche à mémoriser la recherche de la preuve d'insatisfiabilité) mais aussi lors du retour arrière (ce qui suggère que l'on souhaite mémoriser les affectations indépendantes). Dans [20], les auteurs suggèrent que la phase permet de sauvegarder des solutions partielles indépendantes les unes des autres, suggérant que l'on souhaite se focaliser sur la recherche d'un modèle.

Dans cette section, nous proposons quelques pistes de réflexion sur l'amélioration du mécanisme de phase. Comme nous le montrons, nos tentatives pour améliorer la phase dans le cas de la recherche d'un modèle ont échoué. Malgré tout, en spécialisant de manière simple, ce système de sauvegarde, nous avons obtenu des résultats prometteurs sur les instances insatisfiables, donnant un nouveau point de vue de ce mécanisme.

3.1 Une phase pour SAT, une phase pour UNSAT

La phase est un système très simple de sauvegarde permettant de choisir avec quelle polarité on va affecter une variable de décision. Dans le cas d'un problème satisfiable, il est clair que déterminer la bonne phase des variables de décision est aussi difficile que de résoudre ce problème. Dans le cas d'un problème insatisfiable, identifier la bonne polarité est beaucoup moins intuitif. Nous proposons de séparer la sauvegarde de la phase suivant deux évènements bien distincts : (1) lorsque une affectation a été responsable d'un conflit et (2) lorsque elle est indépendante d'un conflit. Le cas (1) intervient durant l'analyse de conflit, sur le dernier niveau de décision, seules les variables qui participent au conflit voient cette phase spéciale (notée `phaseUNSAT`) mise à jour. En effet, toutes les autres variables n'ont pas participé au conflit et n'in-

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = \overline{x_8} \vee \overline{x_7} \vee \overline{x_9}$
$c_8 = \overline{x_7} \vee x_{10}$
$c_9 = \overline{x_7} \vee x_1 \vee x_{12}$



Resolution Steps :
$\beta_1 = res(x_9, \phi_7, c_6) = \overline{x_3} \vee x_8 \vee \overline{x_7} \vee \overline{x_{13}}$
$\beta = res(x_{13}, \beta_1, c_5) = \overline{x_7} \vee \overline{x_3} \vee x_8$

FIGURE 4 – Séquence de décisions (les littéraux de décision sont dans des cercles) suivies de propagations (rectangle). Chaque clause, raison de la propagation d'un littéral, est entre crochets. Un conflit survient au niveau de décision 4. Les étapes de résolution sont explicitées. Le niveau de retour arrière est 2. Le littéral assertif est $\overline{x_7}$.

terviennent donc pas dans le cas (1), mais dans le cas (2). Les variables indépendantes du conflits et qui sont désaffectées lors du retour arrière appartiennent également au cas (2). Toutes ces variables mettent à jour leur polarité dans une nouvelle structure de données nommée `phaseSAT`. Nous notons `polarity`, le mécanisme classique qui est toujours sauvegardé.

Nous illustrons ces deux cas dans l'exemple de la Fig. 4. Nous présentons une simple formule CNF et une séquence de décisions/propagations typique générant un conflit au niveau de décision 4. L'analyse de conflits fait intervenir les variables x_9, x_{13}, x_7 du dernier niveau de décision et x_3 et x_8 qui appartiennent à la clause assertive. Tous ces littéraux mettent à jour leur polarité dans `phaseUNSAT`. A l'opposé, les littéraux x_{10}, x_{12} du dernier niveau de décision et x_2, x_{11} qui sont dé-assignés après retour arrière au niveau 2 ne participent pas à l'analyse de conflit. Il sauvegardent leur polarité dans `phaseSAT`. Toutes ces variables continuent à sauvegarder leur polarité dans `polarity`, comme d'habitude.

Une dernière remarque, ce processus est désactivé au moment du redémarrage (premier conflit après un redémarrage), car nous n'avons aucune idée quant à l'indépendance des affectations dans ce cas précis.

3.2 Comparaison de la phase et du modèle

Nous avons calculé et sauvegardé `phaseSAT` et `phaseUNSAT` pour les 574 problèmes très difficiles (voir la section 4 pour une description de ces instances) sur un temps limite de 10 000 secondes. Nous obtenons un total de 123 instances SAT à analyser. Pour chacune d'entre elles, nous avons comparé le modèle obtenu aux deux mécanismes de sauvegarde proposés (qui peuvent être considérés comme des modèles). Sur la figure 6 et

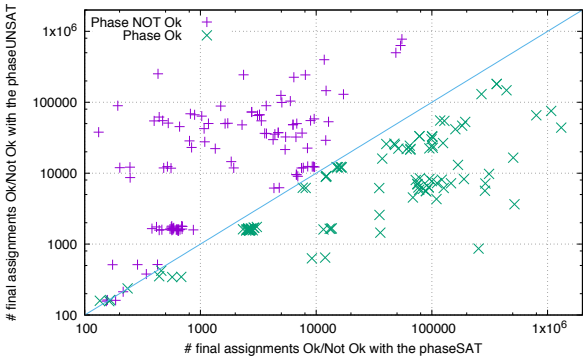


FIGURE 6 – Comparaison sur le nombre d'affectations qui sont en accord/désaccord entre le modèle et la `phaseSAT` ou la `phaseUNSAT`.

pour chaque instance, nous rapportons le nombre de valeurs de `phaseSAT` qui sont les mêmes que la solution trouvée et les comparons au nombre de valeurs de `phaseUNSAT` qui sont les mêmes que la solution trouvée. Et, réalisons la même opération lorsque les valeurs sont en désaccord. Il est clair que tous les points où les phases sont en accord avec le modèle sont en dessous de la diagonale, montrant que la `phaseSAT` est beaucoup plus proche du modèle final que la `phaseUNSAT`. De plus, la `phaseUNSAT` est beaucoup plus en contradiction avec le modèle que la `phaseSAT`.

Notez que certaines valeurs de `phaseSAT`/`phaseUNSAT` peuvent ne pas avoir été initialisées (si la variable n'est jamais intervenue dans l'analyse de conflits par exemple). Dans cette expérience, nous n'avons retenu que les variables pour lesquelles la phase a été initialisée. Cela peut

Solver	SAT	UNS	Total
Glucose	81	151	232
Glucose A	98	162	260
Glucose P	83	150	233
Glucose A+P	**107**	168	**275**
lingeling-ayv	93	158	251
lingeling-baq	77	**185**	262

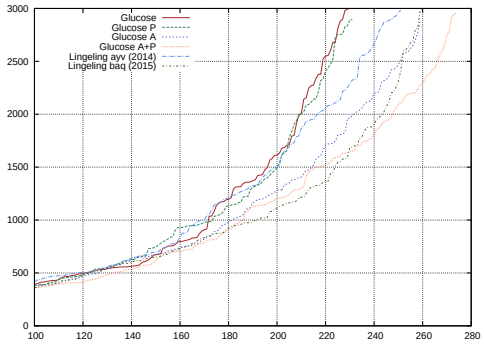


FIGURE 5 – Comparaison de différentes versions de Glucose et de lingeling sur un ensemble de 573 instances difficiles pour Glucose. La table détaille le nombre d'instances SAT/UNSAT résolues, le graphique donne le cactus classique. Le temps est limité à 3000 secondes.

fausser l'impression de proximité des modèles et c'est pourquoi nous avons aussi ajouté les points qui sont en désaccord (ils ne peuvent pas être déduits des points en accord).

3.3 Une tentative pour SAT... qui a échouée

Nous avons voulu prendre en compte la phaseSAT (qui semble plus proche du modèle généré) dans le but d'améliorer Glucose sur les instances SAT. Nous avons testé les deux stratégies suivantes :

— Au cours de la première descente suite à un redémarrage, affecter les variables par rapport à phaseSAT et donc essayer de ce rapprocher d'un modèle potentiel.

— Utiliser la phaseSAT lorsque Glucose décide de retarder un redémarrage [3]. En effet, dans ce cas, le nombre de variables affectées a subitement augmenté, laissant penser que l'on se rapproche d'un modèle.

Ces deux techniques se sont révélées non satisfaisantes. Le nombre d'instances résolues est sensiblement le même. Bien sûr, sur certaines instances satisfiables, le gain a été substentiel. Mais dans la majorité des cas, ce qui a été gagné d'un coté, a été perdu de l'autre. Il faut aussi noter, que ces deux stratégies n'ont quasiment pas eu d'impact sur les instances insatisfiables.

3.4 Une tentative pour UNSAT... qui marche

Nous avons tout d'abord essayer de remplacer complètement la polarity classique par phaseUNSAT. Les résultats ont été désastreux, montrant que garder en mémoire la phaseSAT est important. Néanmoins, phaseUNSAT à une signification précise : elle indique où se trouvent les conflits. Et donc, après un redémarrage, nous avons utilisé phaseUNSAT pour espérer atteindre le plus rapidement possible une zone conflictuelle. Les résultats ont été intéressants et nous en donnons une synthèse dans la section suivante. Mais, comme nous allons le voir, il faut noter que cela est dépendant de la sélection des instances, mais plus incroyable encore, du temps de calcul limité fixé.

4 Experimentations

Dans cette section, nous comparons les différentes versions de Glucose, avec et sans le tuning adaptatif (noté A) et avec ou sans la modification de la phaseUNSAT (note P). Nous avons ajouté dans cette comparaison, la dernière version de lingeling. Nos résultats peuvent paraître disparates au premier abord (avec 3 ensembles de benchmarks et 2 temps limites différents), mais, comme nous l'expliquerons par la suite, nous voulions expliquer plus en détails les résultats surprenants que nous avons obtenus. Dernier détail, notez que la dernière version de lingeling-baq utilise des techniques d'apprentissage pour connaitre les paramètres à utiliser suivant le type de problème.

Les premières expérimentations ont été réalisées sur un sous-ensemble des benchmarks issu des compétitions allant de 2002 à 2013. Nous avons sélectionné 573 benchmarks sur les 2632 et gardé ceux qui n'ont pas été résolus en moins de une minute par GLUCOSE-SYRUP sur 12 coeurs [3]. Notre but, est d'avoir un petit ensemble de problèmes difficiles. Le temps CPU est limité à 3000 secondes. La figure 5 donne un résumé des ré-

3. Plus d'informations sur les instances retenues sont disponibles à http://www.labri.fr/perso/lsimon/sat16

Solver	SAT	UNS	Total
Glucose	104	127	231
Glucose A	**115**	128	243
Glucose P	106	127	233
Glucose A+P	112	129	241
lingeling-ayv	99	143	243
lingeling-baq	106	**146**	**252**

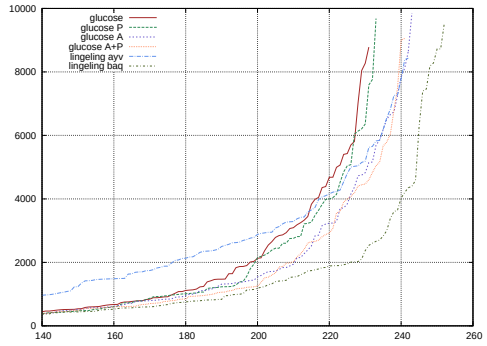


FIGURE 7 – Comparaison de différentes versions de Glucose et de lingeling sur les instances applications de la compétition SAT 2014. La table détaille le nombre d'instances SAT/UNSAT résolues, le graphique donne le cactus classique. Le temps est limité à 10000 secondes.

sultats obtenus. Commençons par comparer Glucose avec ou sans la technique phaseUNSAT. Les résultats sont similaires. Par contre, l'adaptation en cours de recherche est prometteuse, elle permet de résoudre plus d'instances SAT et UNSAT que la version classique. En activant les deux composants (noté A+P) nous obtenons un solveur beaucoup plus efficace, capable de résoudre 43 instances de plus que la version classique de Glucose. Il est étrange de constater que la phaseUNSAT semble inefficace sans l'adaptation. Nous n'avons, pour l'instant, aucune justification.

La comparaison avec lingeling est également surprenante. Cette version est incroyablement efficace sur les instances insatisfiables, mais les performances chutent drastiquement sur les satisfiables. Cette version utilise des techniques d'apprentissage et s'adapte automatiquement, suivant le type de problème. L'apprentissage ayant eu lieu sur les instances de la compétition 2014. Il est possible que l'apprentissage est échoué à classifier correctement les instances satisfiables. En tout cas, notre dernière version de Glucose s'avère être la plus performante.

Nous avons été agréablement surpris d'obtenir d'aussi bons résultats, surtout par rapport à lingeling-baq. Nous avons donc décidé d'aller plus en profondeur dans notre étude en augmentant le timeout et en changeant la base de problèmes servant à nos tests.

Regardons de plus près les résultats obtenus sur les instances de la compétition 2014 en utilisant 10000 secondes comme temps limite. Les résultats sont résumés dans la figure 7. Ici aussi, la technique phaseUNSAT est inefficace sans la stratégie d'adaptation. Par contre, la stratégie adaptative produit les meilleurs résultats. Elle semble donc être indépendante de l'ensemble de benchmarks utilisé. Le fait qu'elle prenne en compte des statistiques sur la recherche et pas sur la syntaxe de la formule peut expliquer cela. Sur l'ensemble des benchmarks de 2014, 210 benchmarks ont un comportement non extrême et environ 40 sont résolus avant 100000 conflits, et donc l'adaptation ne se fait que sur un petit ensemble de benchmarks. 20 d'entre eux, ont des niveaux de décisions faibles et Glucose adapte sa stratégie en conséquence (voir la section 2.3.1). Pour 30 instances, l'adaptation a porté sur le nombre faible de conflits successifs et pour 6 sur un nombre élevé de conflits successifs. Cette adaptation permet de résoudre 12 instances supplémentaires que la version originale. Il est également clair que l'apprentissage de lingeling a été opéré sur ces instances, avec un solveur résultant très robuste. Toutefois, ici aussi, les résultats de lingeling sur les instances satisfiables ne sont pas très impressionnants.

Le dernier ensemble de problèmes vient de la SAT-RACE 2015. Ici aussi nous avons utilisé un temps limité à 10000 secondes. Les résultats, disponibles sur la Figure 8, sont sensiblement les mêmes que pour la compétition 2014. La version adaptative est la meilleure des versions de Glucose et sa combinaison avec la phaseUNSAT semble encore inutile. Le manque de diversité des benchmarks peut expliquer cela. Malgré tout, comme nous pouvons le voir sur le cactus plot, cette combinaison s'avère en fait intéressante lorsque des temps limite plus faibles sont utilisés (voir le cactus de la Fig. 8).

Intuitivement, nous pensons que cette nouvelle phase UNSAT permet d'intensifier la recherche, de manière gloutonne, agressive et surtout plus rapide, en effectuant de meilleurs choix de phase dans la direction des conflits déjà rencontrés. Cependant, après

Solver	SAT	UNS	Total
`Glucose`	148	108	256
`Glucose` A	**154**	108	**262**
`Glucose` P	151	108	259
`Glucose` A+P	150	109	259
`lingeling-ayv`	146	111	257
`lingeling-baq`	146	**115**	261

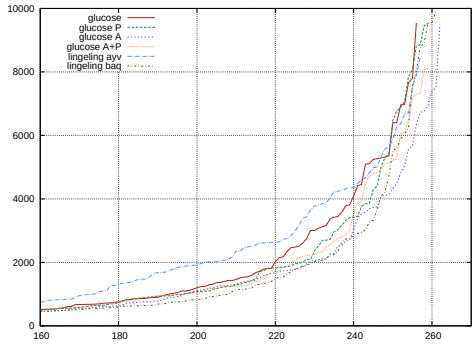


FIGURE 8 – Comparaison de différentes versions de `Glucose` et de `lingeling` sur les instances applications de la compétition SAT 2015. La table détaille le nombre d'instances SAT/UNSAT résolues, le graphique donne le cactus classique. Le temps est limité à 10000 secondes.

un certain temps, l'apprentissage de nouvelles clauses peut certainement corriger les choix de phase moins gloutonnes de l'approche classique et réduire ainsi la différence entre les deux approches.

Le tuning de `lingeling` donne des résultats moins spectaculaires que pour la compétition 2014. Par contre, la version adaptative, via l'apprentissage, est beaucoup plus rapide avec des gains substentiels si on se limite à 3000 secondes.

5 Discussion

Les approches portfolio exploitent le fait que différents solveurs sont efficaces sur différents types de problèmes. Il en existe pour les CSP (Problèmes de Satisfactions de Contraintes) [19, 1], pour SAT [23, 15] ou encore pour les ASP (Answer SET programming) [10]. Après une phase d'entrainement offline, ces solveurs utilisent les techniques d'apprentissage pour choisir le meilleur solveur à exécuter sur le problème considéré.

Plus récemment, de nombreux travaux sur SAT ont porté sur l'adaptation des solveurs suivant le type de benchmark résolu. Par exemple, les méthodes de inprocessing [12] utilisent différentes techniques qui peuvent être désactivées en cours de route si elles s'avèrent inefficaces. De plus, les très bons résultats obtenus par [7] sont surement dus à l'adaptation du solveur suivant les cas (les sources de ce solveur ne sont malheureusement pas disponibles). Durant la SAT-RACE 2015, au moins deux autres solveurs ont utilisé les techniques adaptatives. Le premier est `lingeling` [6]. Cette version utilise les techniques d'apprentissage, plus précisément les algorithmes de type k-proches voisins (KNN) pour classifier un problème parmi l'une des sous-catégories de la compétition 2014 [5]. Si le classificateur valide le choix, les paramètres du solveur sont modifiés en conséquence pour s'adapter à la catégorie en question. L'autre solveur est CRYPTOMINISAT [22]. Il utilise également des techniques d'apprentissage [21] pour sélectionner, après 160000 conflits, une parmi les 13 initialisations possibles du solveur. La phase d'entraînement a été réalisée sur les instances de 2009, 2011 et 2013.

Notre travail est relié à ces approches, mais notre objectif final est différent. Nous souhaitons comprendre de manière profonde le comportement des solveurs SAT et extraire des mesures compréhensibles par l'humain pour les améliorer. Nous n'utilisons donc que 5 mesures pour différencier les instances en différentes familles, la où SATZILLA en utilise plus de 50. Une fois ces cas extrêmes identifiés, nous travaillons sur chaque famille et essayons de comprendre le comportement de notre solveur et de l'adapter en conséquence. Ce n'est donc pas du tout la même philosophie. Dans les années 90, un solveur de types portfolio n'aurait jamais pu trouver des paramètres pour transformer un DPLL en CDCL, par exemple.

6 Conclusion

Il est très difficile de trouver un classement uniforme des solveurs sur des ensembles différents de benchmarks, même lorsque l'on considère un ensemble sélectionné avec soin comme celui des compétitions SAT. Cela peut être problématique car dans la communauté SAT, les résultats empiriques sont très souvent requis lorsque l'on propose une nouvelle idée. Dans cet article, nous montrons qu'un meilleur partitionnement des instances est nécessaire. Nous avons mis en évidence que l'ensemble des benchmarks utilisés lors des compétitions SAT est composés de nombreux problèmes aux comportements extrêmes qui doivent être considérés à part pour pouvoir être résolus efficacement. Un simple

paramétrage des solveurs ne suffira pas, et l'on doit surement les *attaquer* via de méthodes alternatives. Nous proposons dans cet article une classification des problèmes basée sur le comportement durant la recherche plutôt que sur leur origine. Par exemple, des instances nécessitant moins de 30 points de choix avant d'arriver à des conflits ne doivent pas être attaqué avec les solveurs courants, quoiqu'elles représentent initialement.

À partir des observations précédentes, nous avons proposé une simple adaptation de `Glucose`, utilisant 4 mesures aisément compréhensibles. Cette version adaptative s'avère bien plus efficace que la version classique sur les problèmes très difficiles issus des compétitions (allant de 2002 à 2013) et toujours performante sur les dernières d'entre elles. Nous avons également proposé une nouvelle stratégie de choix de la polarité des variables de décisions, qui a également permis un gain substantiel sur les instances considérées.

Références

[1] R. Amadini, M. Gabbrielli, and J. Mauro. Sunny : a lazy portfolio approach for constraint solving. *arXiv preprint arXiv :1311.3353*, 2013.

[2] G. Audemard and L. Simon. Glucose 2.1 : Aggressive, but reactive, clause database management, dynamic restarts (system description). In *Pragmatics of SAT (POS)*, jun 2012.

[3] G. Audemard and L. Simon. Refining restarts strategies for sat and unsat. In *18th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 118–126, 2012.

[4] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *proceedings of IJCAI*, pages 399–404, 2009.

[5] A. Belov, M. Heule, D. Diepold, and M. Järvisalo. The application and the hard combinatorial benchmarks in SAT competition 2014. *SAT Competition*, page 81, 2014.

[6] A. Biere. Lingeling and friends entering the SAT Race 2015.

[7] J. Chen. MiniSAT BCD and abcdSAT : solvers based on blocked clause decomposition. In *SAT Race 2015 solvers description.*

[8] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proceedings of IJCAI*, pages 248–253, 2001.

[9] Niklas Een and Niklas Sörensson. An extensible SAT-solver. In *SAT'03*, pages 502–518, 2003.

[10] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. Schneider, and S. Ziller. A portfolio solver for answer set programming : Preliminary report. In *proceedings of Logic Programming and Nonmonotonic Reasoning*, pages 352–357. 2011.

[11] M. Heule and H. van Maaren. March_dl : Adding adaptive heuristics and a new branching strategy. *JSAT*, 2(1-4) :47–59, 2006.

[12] M. Järvisalo, M. Heule, and A. Biere. Inprocessing rules. In *Proceedings of the International Joint Conference on Automated Reasonning (IJCAR)*, pages 355–370, 2012.

[13] D. LeBerre. Exploiting the real power of unit propagation lookahead. In *Proceedings of SAT*, 2001.

[14] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of Las Vegas algorithms. In *proceedings of ISTCS*, pages 128–133, 1993.

[15] Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. Algorithm portfolios based on cost-sensitive hierarchical clustering. In *Proceedings of IJCAI*, 2013.

[16] J. Marques-Silva and K. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *proceedings of ICCAD'96*, pages 220–227, 1996.

[17] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *proceedings of DAC*, pages 530–535, 2001.

[18] Chanseok Oh. Patching MiniSat to deliver performance of modern SAT solvers, 2015.

[19] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *proceedings of ICAICS*, pages 210–216, 2008.

[20] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In *proceedings of SAT*, pages 294–299, 2007.

[21] J Ross Quinlan. *C4. 5 : programs for machine learning*. Elsevier, 2014.

[22] M. Soos and M. Lindauer. The cryptominisat-4.4 set of solvers at the SAT race 2015.

[23] L. Xu, F. Hutter, H. Hoos, and K Leyton-Brown. Satzilla : Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32 :565–606, 2008.

Branch-and-bound répétitif et programmation par contraintes pour le clustering sous contraintes

Thi-Bich-Hanh Dao[1] **Khanh-Chuong Duong**[1] **Tias Guns**[2] **Christel Vrain**[1]

[1] Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067, Orléans, France

[2] Katholieke Universiteit Leuven, Department of Computer Science, Celestijnenlaan 200A, Leuven, Belgium

{thi-bich-hanh.dao, khanh-chuong.duong, christel.vrain}@univ-orleans.fr tias.guns@cs.kuleuven.be

Résumé

Trouver une partition minimisant le critère de la somme des carrés (WCSS) a été largement étudié et de nombreux algorithmes approchés ou exacts ont été développés. Dans un processus de fouille de données, il est important de pouvoir aussi modéliser des connaissances sur les résultats attendus et récemment a été montré l'intérêt des formalismes déclaratifs pour représenter les contraintes utilisateur. L'algorithme RBBA (pour *repetitive branch-and-bound*) fait partie des algorithmes exacts les plus performants, mais il ne prend pas en compte des contraintes utilisateur. Nous présentons dans ce papier une extension de RBBA permettant d'intégrer de telles contraintes. Elle est réalisée grâce à l'utilisation de la Programmation par Contraintes (PPC) pour les étapes *Branch and Bound*. La PPC permet de facilement modéliser les contraintes. Elle permet de calculer non seulement une borne inférieure (comme suggéré dans RBBA) mais aussi une borne supérieure. Enfin, nous avons développé des mécanismes de propagation adaptés pour profiter davantage des bornes calculées. Des expérimentations sur des bases de données classiques montrent que notre approche dépasse largement, en performance, des méthodes exactes existantes avec contraintes utilisateur.

Abstract

Minimum sum-of-squares clustering (MSSC) is a widely studied task and numerous approximate as well as a number of exact algorithms have been developed for it. Recently the interest of integrating prior knowledge to clustering in Data Mining has been shown, and much attention has gone into incorporating user constraints into clustering algorithms in a generic way. Repetitive Branch-and-Bound Algorithm (RBBA) is one of the best exact approaches for MSSC without user constraints. In this paper we show how RBBA can be extended to incorporate arbitrary user constraints. We develop a framework using Constraint Programming to address constrained MSSC. The key idea is to replace the internal branch-and-bound method by Constraint Programming, and use it to compute tight lower and upper bounds. We also develop a propagator that makes better use of the computed bounds. Experiments on classic datasets show that our approach outperforms the existing state-of-the-art exact approaches.

1 Introduction

Le clustering est une tâche classique de fouille de données avec des applications variées en biologie, chimie, médecine, commerce. Le but est de partitionner un ensemble d'objets en sous-ensembles homogènes, appelés clusters. La notion d'homogénéité est classiquement formulée sous la forme d'un critère, le plus utilisé étant WCSS (Within-Cluster Sum of Squares), défini comme la somme des distances euclidiennes au carré entre chaque objet et le centre du cluster auquel il est affecté. Lorsque ce critère est choisi on parle alors de clustering minimisant la somme des carrés (MSSC). Pour rendre les résultats d'un clustering plus proche des attentes de l'utilisateur, on peut souhaiter intégrer de la connaissance du domaine. Elle est alors souvent modélisée par des contraintes posées sur la partition recherchée.

Le clustering MSSC a été prouvé NP-Difficile [1] et a été étudié dans de nombreux travaux. Beaucoup d'algorithmes sont heuristiques, comme par exemple l'algorithme des k-moyennes, et ne trouvent qu'un optimum local [19]. Ils ont été étendus pour intégrer des contraintes utilisateur, mais ils peuvent échouer pour trouver une solution satisfaisant toutes les contraintes, même s'il en existe une. D'autre part il existe des cadres généraux et déclaratifs fondés sur des outils d'optimisation génériques permettant de modéliser

une grande variété de contraintes et de proposer une solution optimale au problème, dès qu'il existe une solution. Plusieurs cadres ont été proposés, fondé soit sur la Programmation Linéaire en Nombres Entiers avec génération de colonnes [4] ou sur la Programmation par Contraintes (PPC) [9]. Ces cadres sont génériques mais moins efficaces comparativement aux algorithmes dédiés.

Un des meilleurs algorithmes de recherche d'un optimal global pour MSSC sans contraintes utilisateur est RBBA (Repetitive Branch-and-Bound Algorithm) [6]. Notre travail se base sur cet algorithme pour développer un cadre générique avec contraintes utilisateur. Nos contributions sont les suivantes :

- Nous étendons RBBA pour intégrer des contraintes utilisateur.
- Nous présentons un cadre fondé sur la PPC pour réaliser cette extension. L'idée clef est d'utiliser la PPC à chaque itération nécessitant un algorithme de recherche *branch-and-bound*. Nous montrons qu'utiliser la PPC permet de modéliser facilement les contraintes utilisateur et d'obtenir de meilleures bornes inférieures et supérieures.
- Nous développons un mécanisme de propagation permettant une meilleure utilisation des bornes calculées.
- Nous montrons que notre méthode dépasse en performances les méthodes exactes existantes sur des bases de données classiques et que bien qu'étant générique, sa performance reste compétitive avec l'algorithme dédié RBBA dans le cas sans contraintes utilisateur.

Plan. La section 2 introduit des notions de base et la section 3 présente les travaux existants en lien avec ce problème. La section 4 décrit l'algorithme RBBA et l'extension que nous proposons pour intégrer des contraintes utilisateur. Dans la section 5 est présenté le cadre fondé sur la PPC permettant d'implanter cette extension de RBBA. La section 6 est dédiée aux expérimentations permettant de comparer notre méthode aux autres approches exactes existantes. La section 7 discute des perspectives et conclut.

2 Préliminaires

Considérons un ensemble $\mathcal{O}$ de N objets dans un espace euclidien. Soit d la distance euclidienne ($d(o,o') = ||o - o'||$). Le clustering MSSC (Minimum Sum-of-Squares Clustering) vise à trouver une partition Δ des objets en K clusters $C_1, ..., C_K$ telle que : (1) $\forall k \in \{1,..,K\}$, $C_k \neq \emptyset$, (2) $\bigcup_k C_k = \mathcal{O}$, (3) $\forall k \neq k'$, $C_k \cap C_{k'} = \emptyset$ et (4) la somme des carrés intra-clusters (WCSS) est minimisée, où WCSS est définie par :

$$WCSS(\Delta) = \sum_{k\in\{1,..,K\}} \sum_{o\in C_k} d(o, m_k)^2$$

où pour chaque $k \in [1, K]$, m_k est le centre du cluster C_k. De façon équivalente [12, 14] :

$$WCSS(\Delta) = \sum_{k\in\{1,..,K\}} \frac{\frac{1}{2}\sum_{o,o'\in C_k} d(o,o')^2}{|C_k|}$$

A noter que d'autres critères existent pour le clustering, comme par exemple

- minimiser la somme des dissimilarités intra-cluster : $\sum_{k\in\{1,..,K\}} \sum_{o,o'\in C_k} d(o,o')^2$,
- minimiser le diamètre maximal des clusters : $\max_{k\in\{1,..,K\}} \max_{o,o'\in C_k} d(o,o')$),
- maximiser la séparation minimale entre clusters : $\min_{k,k'\in\{1,..,K\},k\neq k'} \min_{o\in C_k,o'\in C_{k'}} d(o,o')$.

Un utilisateur peut aussi disposer de connaissances préalables au clustering, ou souhaiter imposer des contraintes sur les clusters. Par exemple, il peut connaître des étiquettes sur un sous-ensemble des objets ou une borne supérieure sur le nombre d'objets de chaque cluster. De telles connaissances sont intégrées dans le processus de clustering par des contraintes utilisateur qui doivent être satisfaites. On distingue en général les *contraintes sur les instances*, introduite dans [21], certainement les plus utilisées et spécifiant des contraintes sur des paires d'objets, des *contraintes sur les clusters*. Les *contraintes sur les instances* sont soit des contraintes *must-link* (ML) ou des contraintes *cannot-link* (CL) spécifiant que deux objets doivent être, ou resp. ne peuvent pas être, dans le même cluster. On distingue aussi plusieurs types de contraintes sur les clusters, les plus populaires étant les suivantes.

- Une contrainte de diamètre fixe une borne supérieure γ sur le diamètre des clusters : $\forall k \in [1, K]$, $\forall o, o' \in C_k$, $d(o,o') \leq \gamma$. Cette contrainte peut être exprimée par des contraintes CL : chaque paire d'objets o, o' à une distance strictement supérieure à γ doivent être dans des clusters différents ($CL(o,o')$).
- Une contrainte de marge fixe une borne inférieure δ sur la distance séparant les clusters : $\forall k \neq k' \in [1, K]$, $\forall o \in C_k$, $\forall o' \in C_{k'}$, $d(o,o') \geq \delta$. Cette contrainte peut être exprimée par des contraintes ML : chaque paire d'objets o, o' à une distance strictement inférieure à δ doivent être dans les mêmes clusters ($ML(o,o')$).
- Une contrainte de densité exprime que chaque point doit avoir dans un voisinage de rayon ϵ au moins m points appartenant au même cluster que lui : $\forall k \in [1, K]$, $\forall o \in C_k$, $\exists o_1, .., o_m \in C_k \setminus \{o\}$, $d(o, o_i) \leq \epsilon$.

— Une contrainte de taille minimale (resp. maximale) impose à chaque cluster de posséder au moins (resp. au plus) un nombre α (resp. β) fixé d'objets : $\forall k \in [1, K]$, $|C_k| \geq \alpha$ (ou $|C_k| \leq \beta$, resp.).

3 Autres travaux

Le clustering minimisant la somme des carrés (MSSC) sous contraintes a été étudié avec des approches heuristiques et exactes. Parmi les approches heuristiques, l'algorithme des k-moyennes trouve un optimum local dans le cas sans contraintes utilisateur (voir [19]). Il a été étendu aux contraintes d'instances avec COP-kmeans [22] ou LCVQE [18]. Cependant lorsque le nombre de contraintes augmente, de tels algorithmes échouent souvent à trouver une solution satisfaisant toutes les contraintes même s'il en existe une, ou ne trouve qu'une solution ne satisfaisant pas toutes les contraintes.

Les approches exactes pour MSSC sans contraintes utilisateur utilisent des algorithmes de recherche branch-and-bound [16, 5, 6], la programmation dynamique [15, 20], la programmation linéaire en nombres entiers (ILP) et la génération de colonnes [11, 3], la méthode des plans sécants [23] ou un programme semi-défini branch-and-cut [2]. Il existe peu de méthodes exactes pour MSSC intégrant des contraintes utilisateur [4, 9]. Elles sont fondées sur des outils d'optimisation génériques, de manière à modéliser plusieurs types de contraintes utilisateur. Etendant [3], un cadre fondé sur ILP et la génération de colonnes a été proposé dans [4]. Un cadre générique fondé sur la PPC a été développé dans [9], avec une contrainte globale pour calculer et élaguer l'espace de recherche pour le critère WCSS de MSSC.

Il existe aussi d'autres modélisations du clustering sous contraintes avec des fonctions objectif différentes. [17] propose une approche fondée sur ILP ; un ensemble de clusters candidats doit être donné en entrées et plusieurs critères peuvent être considérés pour choisir le meilleur sous-ensemble de clusters. Une approche fondée sur SAT a aussi été proposée pour le clustering sous contraintes avec les critères de diamètre ou de séparation [10]. Un cadre fondé sur la PPC a été développé dans [7, 8] incluant comme critères d'optimisation le diamètre, la séparation, la somme des dissimilarités intra-cluster et des contraintes utilisateur.

Notre travail étend l'algorithme RBBA [6], qui trouve un optimum global pour MSSC sans contraintes utilisateur. Nous montrons que la méthode que nous proposons peut être combinée avec un cadre en PPC, donnant une méthode efficace permettant d'intégrer facilement des contraintes utilisateur.

4 Extension de RBBA aux contraintes utilisateur

Soit $\mathcal{O}$ un ensemble de N points, et soit Δ une partition de $\mathcal{O}$ en *au plus* K clusters. Pour chaque sous-ensemble S de $\mathcal{O}$, soit Δ_S la projection de Δ sur les objets de S et $WCSS(\Delta_S)$ la valeur WCSS de Δ_S. Soit $WCSS^*(S) = \min_\Delta(WCSS(\Delta_S))$. Notons que dans Δ_S certains clusters de Δ peuvent être vides.

4.1 Inégalité de borne inférieure sans contraintes utilisateur

L'algorithme RBBA et l'extension que nous proposons reposent sur le résultat suivant [16]. Soit S un sous-ensemble de $\mathcal{O}$ et soient S_1 et S_2 tels que $S = S_1 \cup S_2$ et $S_1 \cap S_2 = \emptyset$. Nous avons alors :

$$WCSS(\Delta_S) \geq WCSS(\Delta_{S_1}) + WCSS(\Delta_{S_2}) \quad (1)$$

Puisque $WCSS^*(S_2)$ est la plus petite valeur de WCSS pour toutes les partitions de S_2 en au plus K clusters, nous avons :

$$WCSS(\Delta_{S_2}) \geq WCSS^*(S_2) \quad (2)$$

et donc :

$$WCSS(\Delta_S) \geq WCSS(\Delta_{S_1}) + WCSS^*(S_2) \quad (3)$$

L'inégalité (3) peut être utilisée pour trouver une partition optimale de S comme suit. Supposons que nous avons déjà construit une partition de S, nous donnant ainsi une borne supérieure pour $WCSS^*(S)$, que nous avons une solution partielle Δ_{S_1} et que nous connaissons une solution optimale de $WCSS^*(S_2)$. Alors si $WCSS(\Delta_{S_1}) + WCSS^*(S_2)$ est plus grand que la borne supérieure actuelle, l'espace de recherche peut être élagué.

4.2 Algorithme RBBA

L'algorithme RBBA (Repetitive Branch-and-Bound Algorithm) [6] est décrit dans l'algorithme 1.

Dans cet algorithme, $\mathcal{O}_n$ représente les n derniers points (de $N - n + 1$ à N), Δ_n une partition de $\mathcal{O}_n$ en au plus K clusters, Δ_n^* une partition optimale de $\mathcal{O}_n$ en au plus K clusters et W_n le score optimal pour WCSS ($W_n = WCSS(\Delta_n^*)$). Tout d'abord, les points de $\mathcal{O}$ sont ordonnés *Ordering($\mathcal{O}$)*. Plusieurs heuristiques peuvent être utilisées, elles seront présentées en sous-section 4.5. Nous supposons que les points sont alors nommés par leur index $i \in [1, N]$. *Init($\mathcal{O}_K$)* crée ensuite Δ_K^* en mettant chaque point seul dans un cluster, et par conséquent $W_K = 0$.

Greedy_Extension($\mathcal{O}_n, \Delta_{n-1}^*$) est un algorithme glouton pour construire une partition Δ_n de $\mathcal{O}_n$, en

Algorithm 1: RBBA

```
Ordering(O)
O_K ← {N − K + 1, ..., N}
Δ*_K ← Init(O_K)
W_K ← 0
for n = K + 1 to N do
    O_n ← O_{n−1} ∪ {N − n + 1}
    Δ_n ← Greedy_Extension(O_n, Δ*_{n−1})
    U_n ← WCSS(Δ_n)
    Δ*_n ← BaB_Search(O_n, U_n, W)
    W_n ← WCSS(Δ*_n)
```

ajoutant le nouveau point à la meilleure partition précédente Δ^*_{n-1} de manière à augmenter le moins possible WCSS. La valeur $WCSS(\Delta_n)$ constitue une borne supérieure U_n de $WCSS(\Delta^*_n)$.

BaB_Search$(\mathcal{O}_n, U_n, W)$ est un algorithme de type branch-and-bound qui cherche une partition optimale Δ^*_n de l'ensemble $\mathcal{O}_n$, en utilisant U_n comme borne supérieure et en exploitant l'inégalité (3) avec les valeurs W_i $(i < n)$ comme bornes inférieures. Soit $m = N - n + 1$ le nouveau point ajouté à cette étape. Dans l'algorithme branch-and-bound, les points de $\mathcal{O}_n$ sont traités dans l'ordre $m, m+1, \ldots, N$. Considérons une étape où un point p $(m \leq p < N)$ est affecté à un cluster. Soit S_1 l'ensemble de points $\{m, ..., p\}$ et S_2 l'ensemble $\{p+1, ..., N\}$. Tous les points de S_1 ont déjà été instanciés et ceux de S_2 ne le sont pas encore. Soit Δ la partition en construction. $WCSS(\Delta_{S_1})$ est connu et $WCSS^*(S_2)$ a été calculé dans une itération précédente de RBBA et stocké dans $W_{|S_2|}$. Soit U_n la borne supérieure actuelle. L'inégalité (3) est utilisée et si $WCSS(\Delta_{S_1}) + WCSS^*(S_2) \geq U_n$, il ne sera pas possible d'étendre Δ_{S_1} en une solution avec une valeur WCSS meilleure que U_n. En conséquence, BaB_Search ne cherche pas à étendre Δ_{S_1} et l'espace de recherche est élagué. Quand $p = N$, la partition Δ est complète et U_n est fixé à $WCSS(\Delta)$. Si l'espace de recherche est complètement exploré, la dernière partition Δ trouvée est une solution optimale.

Les solutions optimales, calculées à des étapes précédentes sont utilisées comme bornes inférieures pour élaguer la recherche. Les bornes supérieures fournies par l'algorithme glouton sont aussi importantes, elles sont en général assez proches de la solution optimale, elles correspondent même souvent à la solution optimale. Grâce à ces bornes, même si RBBA exécute N fois une recherche branch-and-bound, il est néanmoins le meilleur algorithme exact pour le clustering minimisant la somme des carrés.

4.3 Inégalités de borne inférieure avec contraintes utilisateur

Considérons un ensemble de contraintes utilisateur $\mathcal{C}$ sur $\mathcal{O}$ et étudions sous quelles conditions l'inégalité (3) reste valide. Notons $WCSS^*(\mathcal{O}, \mathcal{C})$ la valeur optimale de WCSS pour $\mathcal{O}$ sous les contraintes $\mathcal{C}$. L'espace de recherche, dénoté par $\mathcal{S}(\mathcal{O}, \mathcal{C})$ est l'ensemble de toutes les partitions de $\mathcal{O}$ satisfaisant $\mathcal{C}$. *Plus généralement, soit S un sous-ensemble de $\mathcal{O}$ et C un ensemble de contraintes sur S, $\mathcal{S}(S, C)$ représente l'ensemble des partitions de S satisfaisant C.*

L'inégalité (1) compare une partition et ses projections sur deux sous-ensembles complémentaires. Aucune condition n'est posée sur les partitions, elle reste donc vraie, même si les partitions satisfont des contraintes. L'inégalité (3) repose sur les inégalités (1) et (2). Lorsque l'on considère des contraintes utilisateur, deux questions se posent :

1. Etant donné un ensemble C de contraintes sur S, quel est l'ensemble de contraintes, noté C_{S_2}, posé sur S_2 ?
2. $WCSS^*(S_2, \mathcal{C}_{S_2})$ représente la valeur optimale de WCSS parmi les partitions de $\mathcal{S}(S_2, C_{S_2})$. Est-ce que l'inégalité (2) est encore valide ? Autrement dit, si Δ_S satisfait C, est-ce que sa projection sur S_2 satisfait C_{S_2} ?

Notons que si C_1 et C_2 sont deux ensembles de contraintes telles que $C_1 \subseteq C_2$, alors $\mathcal{S}(S, C_1) \supseteq \mathcal{S}(S, C_2)$ et donc $WCSS^*(S, C_1) \leq WCSS^*(S, C_2)$. Nous distinguons deux types de contraintes :

— les contraintes d'instances, telles que les contraintes ML ou CL posées sur des paires d'objets,

— les contraintes sur les clusters, telles que la capacité maximale ou minimale des clusters. Rappelons que certaines contraintes sur les clusters comme le diamètre ou la marge peuvent être traduites en contraintes CL (resp. ML).

Soit C un ensemble de contraintes sur S, nous notons C_{S_2} l'ensemble des contraintes d'instances de C posées sur les objets de S_2. Si une partition Δ satisfait un ensemble C de contraintes, alors sa projection sur S_2 satisfait les contraintes C_{S_2}. Donc :

$$WCSS(\Delta_S, \mathcal{C}) \geq WCSS(\Delta_{S_1}) + WCSS^*(S_2, C_{S_2}) \tag{4}$$

Concernant les contraintes sur les clusters, nous introduisons la notion de contrainte anti-monotone comme suit. Une contrainte c est dite *anti-monotone* si lorsqu'elle est satisfaite par une partition Δ_S, alors elle est satisfaite par toutes les projections Δ_{S_2} sur des sous-ensembles S_2 de S. Autrement dit, soit v_c la fonction booléenne qui teste si c est satisfaite sur

une partition. Une contrainte anti-monotone satisfait la propriété suivante : si Δ est une partition sur S et si $S_2 \subseteq S$ alors $v_c(\Delta_{S_2}) \geq v_c(\Delta)$. Par exemple une contrainte de capacité maximale est anti-monotone, alors qu'une contrainte de capacité minimale ne l'est pas. Soit C_a les contraintes anti-monotones de C. Alors, nous avons :

$$WCSS(\Delta_S, C) \geq WCSS(\Delta_{S_1}) + WCSS^*(S_2, C_{S_2} \cup C_a) \quad (5)$$

4.4 RBBA avec contraintes utilisateur

Soit $\mathcal{C}$ l'ensemble des contraintes utilisateur sur $\mathcal{O}$. Supposons que $\mathcal{C}$ soit satisfiable, ie. il existe une partition Δ de $\mathcal{O}$ qui satisfait $\mathcal{C}$. L'extension de RBBA intégrant les contraintes utilisateur est présentée dans l'algorithme 2.

Après avoir ordonné les points, l'algorithme 2 considère l'ensemble $\mathcal{O}_K = \{N-K+1, \ldots, N\}$ et construit une partition Δ_K en *au plus* K clusters comme suit. Soit $\mathcal{C}_K = \mathcal{C}_{\mathcal{O}_k}$, l'ensemble des contraintes d'instances sur $\mathcal{O}_k$. Pour tout point $i \in \mathcal{O}_K$, s'il existe une contrainte ML sur i avec un autre point $j \in \mathcal{O}_K$ déjà affecté, alors i est affecté au même cluster que j, sinon i est mis dans un nouveau cluster. Puisque $\mathcal{C}$ est satisfiable sur $\mathcal{O}$, la partition Δ_K satisfait $\mathcal{C}_K$. Puisque chaque point est mis seul dans un cluster dès lors qu'il n'existe pas de contraintes ML dans $\mathcal{O}_K$ portant dessus, Δ_K est un clustering optimal Δ_K^* pour $\mathcal{O}_K$ et $\mathcal{C}_K$.

A chaque étape n, soit $\mathcal{O}_n$ l'ensemble composé des n derniers points, l'algorithme 2 cherche dans l'espace des solutions $\mathcal{S}(\mathcal{O}_n, \mathcal{C}_n)$, ie. l'ensemble des partitions de $\mathcal{O}_n$ qui satisfont $\mathcal{C}_n$. *Feasible_Extension* essaie d'étendre la meilleure partition obtenue à l'étape précédente Δ_{n-1}^* en une partition Δ_n de $\mathcal{O}_n$ satisfaisant $\mathcal{C}_n$. Si une telle extension Δ_n existe, $WCSS(\Delta_n)$ est une borne supérieure pour $WCSS(\Delta_n^*)$. Sinon, la borne supérieure est fixée à l'∞. *Constrained_BaB*$(\mathcal{O}_n, \mathcal{C}_n, U_n, W)$ effectue une recherche branch-and-bound pour trouver une partition optimale parmi toutes les partitions satisfaisant $\mathcal{C}_n$. Il utilise U_n comme borne supérieure initiale et W comme bornes inférieures, de la même manière que *BAB_Search* dans l'algorithme 1.

Constrained_BaB$(\mathcal{O}_n, \mathcal{C}_n, U_n, W)$ est un algorithme de recherche branch-and-bound, sous contraintes, i.e., il prend en compte les contraintes $\mathcal{C}_n$ pendant la recherche d'une solution optimale. Il existe plusieurs choix possibles pour l'ensemble de contraintes $\mathcal{C}_n$ passé en argument de la méthode. On peut utiliser l'ensemble $\mathcal{C}_{\mathcal{O}_n}$ ou $\mathcal{C}_{\mathcal{O}_n} \cup \mathcal{C}_a$ où $\mathcal{C}_a$ est l'ensemble des contraintes anti-monotones sur les clusters. Remarquons que plus le nombre de contraintes considéré est important, plus serrée sera la borne pour les étapes suivantes. Nous utilisons la programmation par contraintes pour réaliser *Constrained_BaB*, comme expliqué dans la section suivante.

Algorithm 2: Extended RBBA

```
1  Ordering(O)
2  O_K ← {N − K + 1, ..., N} ; C_K ← C_{O_K}
3  Δ_K ← Init(O_K)
4  W_K ← WCSS(Δ_K)
5  for n = K + 1 to N do
6      O_n ← O_{n−1} ∪ {N − n + 1}
7      Δ_n ← Feasible-Extension(O_n, C_n, Δ*_{n−1})
8      if Δ_n exists then
9          U_n ← WCSS(Δ_n)
10     else
11         U_n ← ∞
12     Δ*_n ← Constrained_BaB(O_n, C_n, U_n, W)
13     W_n ← WCSS(Δ*_n)
```

4.5 Ordonnancement des points

Les algorithmes 1 et 2 commencent par ordonner les points et répètent les étapes branch-and-bound en suivant cet ordre. Dans RBBA [6], une heuristique de séparation des plus proches voisins (nearest-neighbor separation NNS) est utilisée : à chaque étape de l'ordonnancement, les deux points qui ont la distance la plus faible sont retirés et placés aux extrémités opposées dans l'ordonnancement. Elle a pour but de mettre des points *faciles à classer*, car proches d'un point déjà classé, vers la fin du processus.

L'ordre que nous utilisons est fondé sur l'algorithme FPF (furthest-point-first) [13]. Il commence par choisir le point le plus éloigné des autres points, le marquant comme la première *tête* et associe tous les points à cette première tête. A chaque itération le point i qui est le plus loin de sa tête est marqué comme une nouvelle tête et les points non encore marqués qui sont plus proches de lui que de leur tête lui sont alors associés. L'algorithme s'arrête lorsque tous les points sont marqués. L'ordre choisi est l'ordre de marquage des points. Cet ordonnancement tend à mettre les points loin les uns des autres au début, visant à traiter les points perturbateurs au début du processus.

5 Un cadre en programmation par contraintes

Nous présentons un cadre à base de programmation par contraintes (PPC) pour réaliser l'algorithme 2.

La PPC est utilisée à la fois pour la recherche par branch-and-bound (*Constrained_Bab*) et pour trouver un clustering de départ (*Feasible_Extension*). Elle est également utilisée pour trouver de meilleures bornes inférieures et supérieures.

5.1 Un modèle en PPC pour Constrained_BaB

Etant donnés un ensemble de n points $\mathcal{O}_n$, un ensemble de contraintes $\mathcal{C}_n$, une borne supérieure U_n de la somme des carrés WCSS sur $\mathcal{O}_n$ et W les valeurs de WCSS calculées dans les itérations précédentes, *Constrained_BaB*$(\mathcal{O}_n, \mathcal{C}_n, U_n, W)$ dans l'algorithme 2 a pour objectif de trouver un clustering Δ_n^* sur $\mathcal{O}_n$, qui satisfait toutes les contraintes de $\mathcal{C}_n$ et qui minimise la somme des carrées WCSS.

Afin de construire un modèle en PPC pour cette tâche, nous étendons le modèle pour le clustering sous contraintes développé en [8]. Afin de définir une répartition de n points en K clusters, n variables entières $G_1, \ldots, G_n$ sont utilisées, avec $Dom(G_i) = \{1, .., K\}$. L'affectation $G_i = k$ signifie que le point i est mis dans le cluster k. Une affectation complète définit donc une partition, mais une partition peut correspondre à plusieurs affectations différentes (par exemple en permutant les indices des clusters). Afin de casser ces symétries entre partitions, la contrainte *precede*$([G_1, .., G_n], [1, .., K])$ est utilisée. Elle impose que le point 1 soit dans le cluster 1 et un point i ne peut être dans un cluster k que si le cluster $k-1$ contient un point j avec $j < i$. Tous les types de contraintes utilisateur introduits en Section 2 peuvent être exprimés en utilisant des contraintes prédéfinies de PPC [8].

Pour présenter la somme des carrés du clustering défini par les variables G, nous introduisons une variable V de type virgule flottante. Le domaine de V est l'intervalle $[V.lb, V.ub)$, avec initialement $V.lb = 0$ et $V.ub = U_n$. La relation que V représente la somme des carrés du clustering défini par (le domaine) de G est renforcée par une nouvelle contrainte globale *sumSquares*(G, V, d, W).

5.2 Une nouvelle contrainte somme des carrés

La contrainte *sumSquares*(G, V, d, W) exprime que V est la somme des carrés du clustering défini par G. Rappelons que d est la mesure de distance et W est un tableau contenant les valeurs WCSS calculées dans les itérations précédentes de RBBA. Le filtrage pour cette contrainte est présenté dans l'algorithme 3.

Suivant le principe de RBBA, pendant la recherche par *branch-and-bound*, les variables $G_1, \ldots, G_n$ sont affectées dans l'ordre croissant de leur indice. Une instanciation partielle est donc telle qu'il existe un indice p $(1 \leq p < n)$ où $G_1, \ldots, G_p$ sont toutes affectées et G_{p+1} ne l'est pas. Les variables $G_{p+1}, \ldots, G_n$ correspondent aux $n-p$ derniers points de $\mathcal{O}$. Puisque RBBA itère en augmentant la valeur de n, les cas correspondant aux $n-p$ derniers points ont déjà été considérés dans une étape précédente. La valeur minimale WCSS de ces points est stockée dans W_{n-p}.

L'algorithme 3 assure d'abord la consistance de borne pour la variable V en calculant une borne inférieure. Les valeurs $sum[k]$ et $size[k]$ représentent respectivement la somme des distances au carré des paires de points dans le cluster k et le nombre de points dans ce cluster. La valeur V_1 représente la valeur WCSS du clustering partiel formé par les p variables déjà affectées. Puisque W_{n-p} représente la valeur WCSS minimale pour les derniers $n-p$ points, d'après les inégalités (4) et (5), $V_1 + W_{n-p}$ est une borne inférieure pour V (ligne 12). Puisque $V.lb \leq V < V.ub$, un échec sera invoqué si $V_1 + W_{n-p} \geq V.ub$ (ligne 11).

L'algorithme 3 exploite aussi W pour filtrer le domaine de la variable G_{p+1}. Chaque valeur $s[k]$ représente la contribution du point $p+1$ dans le cas où il serait affecté au cluster k. Pour chaque $k \in Dom(G_{p+1})$, V_1' est la valeur révisée de V_1 si le point $p+1$ est affecté au cluster k. V_1' représente aussi la valeur WCSS du clustering partiel formé par les $p+1$ premiers points. Du fait que W_{n-p-1} représente la valeur WCSS minimale des derniers $n-p-1$ points, d'après l'inégalité (4), si $V_1' + W_{n-p-1} \geq V.ub$ un échec sera invoqué. Dans ce cas, le point $p+1$ ne pourra pas être mis dans le cluster k, la valeur k est donc supprimée du $Dom(G_{p+1})$.

La complexité de l'algorithme est $O(p^2)$, à cause du calcul de *sum* et *size*. Elle peut se réduire $O(p)$ si les tableaux *sum* et *size* sont stockés et calculés de façon incrémentale sur les différentes propagations.

5.3 Modèles local vs. complet

Soit $\mathcal{C}$ l'ensemble des contraintes utilisateur sur l'ensemble total de points $\mathcal{O}$. Les contraintes de $\mathcal{C}$ peuvent être des contraintes d'instance (must-link, cannot-link) ou de clusters (taille, densité, etc.). A chaque itération n, *Constrained_BaB* trouve un clustering qui minimise la valeur WCSS et qui satisfait l'ensemble de contraintes $\mathcal{C}_n$. Nous proposons deux méthodes différentes pour définir l'ensemble $\mathcal{C}_n$ et donc pour spécifier le modèle en Sous-section 5.1.

Modèle local A l'étape n, soit $\mathcal{O}_n$ l'ensemble des n points considérés. Le modèle local porte sur les n variables $G_1, \ldots, G_n$ correspondant aux n points. L'ensemble $\mathcal{C}_n$ est $\mathcal{C}_{\mathcal{O}_n}$, ie. l'ensemble des contraintes d'instance sur les points de $\mathcal{O}_n$. Lorsque $n = N$, on a $\mathcal{C}_n = \mathcal{C}$.

Algorithm 3: Filtering of *sumSquares*(G, V, d, W)

```
input: G_1, ..., G_p assigned, G_{p+1} unassigned
// computation of lower bound for V
for k = 1 to K do
    sum[k] ← 0 ; size[k] ← 0 ; s[k] ← 0
for i = 1 to p do
    k ← G_i.val() ; size[k] ← size[k] + 1
    for j = i + 1 to p do
        if G_j.val() == k then
            sum[k] ← sum[k] + d(i, j)^2
V_1 ← 0
for k = 1 to K do
    V_1 ← V_1 + sum[k]/size[k]
if V_1 + W_{n-p} ≥ V.ub then return Failure ;
else V.lb ← max(V.lb, V_1 + W_{n-p}) ;
// look ahead to filter Dom(G_{p+1})
for i = 1 to p do
    s[G_i.val()] ← s[G_i.val()] + d(i, p + 1)^2
foreach k in Dom(G_{p+1}) do
    V'_1 ← V_1 − sum[k]/size[k] + (sum[k] +
    s[k])/(size[k] + 1)
    if V'_1 + W_{n-p-1} ≥ V.ub then
        remove k from Dom(G_{p+1})
```

Modèle complet A chaque étape $n \leq N$, *Constrained_BaB* utilise le modèle complet qui est sur N variables $G_1, \ldots, G_N$ correspondant aux N points. L'ensemble de contraintes $\mathcal{C}_n$ est toujours $\mathcal{C}$. Cependant, puisque nous cherchons un clustering optimal sur les derniers n points, la contrainte *sumSquares* est définie sur V et seulement sur les n dernières variables $G_{N-n+1}, \ldots, G_N$. Le branchement est effectué seulement sur ces n variables.

L'intérêt du modèle complet est qu'il peut permettre de détecter plus tôt des cas inconsistants. Par exemple considérons 3 points a, b, c, $(N = 3)$, $K = 2$ et deux contraintes cannot-link $CL(a, b)$ et $CL(a, c)$. A l'étape $n = 2$, les deux derniers points sont considérés $(\mathcal{O}_2 = \{b, c\})$. Le modèle local est défini sur G_b et G_c et n'a pas de contraintes. Il trouve donc comme solution un clustering Δ_2^* où chaque point est dans un cluster $(WCSS(\Delta_2^*) = 0)$. Ce clustering ne peut pas être étendu à l'étape suivante, où $\mathcal{O}_3 = \{a, b, c\}$ à cause des contraintes CL. Le modèle complet quant à lui, est défini toujours sur 3 variables G_a, G_b, G_c avec deux contraintes $G_a \neq G_b$ et $G_a \neq G_c$. A l'étape $n = 2$, même si seulement deux variables G_b et G_c sont instanciées, l'existence de G_a dans le modèle empêche que b et c soient dans deux clusters différents, car sinon $Dom(G_a) = \emptyset$. Le modèle complet peut donc trouver une borne inférieure plus élevée, meilleure car plus réaliste, pour la valeur WCSS des étapes suivantes.

5.4 Construire un clustering de départ

Dans l'algorithme 1, à chaque étape, RBBA commence par construire un clustering de départ Δ_n dont la valeur WCSS constitue une borne supérieure pour la recherche *branch-and-bound* (ligne 7). Sans contraintes utilisateur, Δ_n est créé en ajoutant le nouveau point au clustering solution de l'étape précédente Δ_{n-1}^* de sorte que la valeur WCSS augmente le moins possible. Cette extension donne souvent une borne supérieure assez proche, parfois même la meilleure solution. Pour *Feasible_Extension*$(\mathcal{O}_n, \mathcal{C}_n, \Delta_{n-1}^*)$ dans l'algorithme 2, l'extension devient plus subtile car Δ_n doit satisfaire toutes les contraintes utilisateur $\mathcal{C}_n$.

Feasible_Extension$(\mathcal{O}_n, \mathcal{C}_n, \Delta_{n-1}^*)$ doit trouver un clustering satisfaisant $\mathcal{C}_n$ rapidement. Afin d'équilibrer la qualité de la solution et le temps de calcul, le modèle en sous-section 5.1 est utilisé dans un cas restreint : les dernières $n - 1$ variables $G_2, \ldots, G_n$ sont affectées en suivant le clustering Δ_{n-1}^*. Le branchement est seulement sur G_1, ce qui simule une recherche gloutonne. S'il existe un clustering qui étend Δ_{n-1}^* et qui satisfait $\mathcal{C}_n$, le modèle va trouver le meilleur parmi toutes les possibilités. Si une telle extension n'existe pas, la borne supérieure pour la valeur WCSS est ∞.

5.5 Contraintes must-link

Les contraintes must-link, si elles existent, regroupent les points concernés dans un même cluster. Afin d'exploiter davantage ces contraintes, la fermeture transitive de ces contraintes est d'abord calculée. Cette fermeture transitive définit un ensemble de ML-blocs, et au lieu de chercher un clustering sur les points initiaux, nous cherchons un clustering sur les ML-blocs. Etant donnés N points, supposons que les contraintes must-link créent M blocs $(M \leq N)$. La distance entre deux blocs b_i, b_j est définie par $d(b_i, b_j) = \sqrt{\sum_{o \in b_i, o' \in b_j} d(o, o')^2}$. Chaque bloc b_i a son poids $w(i) = \sum_{o,o' \in b_i} d(o, o')^2/2$ et sa taille $s(i)$ qui est le nombre de point initiaux dans le bloc. Un bloc b_i qui contient un seul point initial aura $w(i) = 0$ et $s(i) = 1$. Les contraintes d'instance qui restent à satisfaire sont seulement des contraintes cannot-link. Une contrainte cannot-link est définie sur deux blocs b_i, b_j s'il existe une contrainte cannot-link sur deux points o, o' tels que $o \in b_i$ et $o' \in b_j$.

Utiliser des blocs signifie que dans le modèle de la sous-section 5.1, chaque variable G_i correspond à un bloc b_i. Les contraintes de cluster peuvent être exprimées sur des blocs. Par exemple, une contrainte

de taille minimale indique que chaque cluster doit avoir au moins α points initiaux. Pour exprimer cette contrainte, nous définissons un tableau T, qui contient des variables G_i chacune répétée $s(i)$ fois. T est donc de longueur N et la contrainte de taille minimale est exprimée par $|\{j \in \{1,\ldots,N\} \mid T_j = k\}| \geq \alpha$ pour $k \in [1, K]$.

6 Expérimentations

Nous comparons dans les expérimentations CPRBBA avec les autres approches exactes RBBA [1], CPClustering 2.1 [2] et CCCG-0.5.1 [3]. Les expérimentations sont réalisées dans les deux cas avec ou sans contraintes utilisateur. Nous avons développé CPRBBA et la contrainte *sumSquares* en utilisant la bibliothèque de PPC Gecode version 4.3.3. Des bases de données classiques du dépôt UCI [4] sont utilisées, avec le nombre réel de classes si ce nombre est connu. Les expérimentations sont effectuées sur un processeur Intel Xeon E3-1225 CPUs sous Ubuntu 14.04 ; des limites sont fixées en temps (30 minutes) et en mémoire (4 Go, qui n'est jamais atteinte).

Ordre des points. L'ordre des points définit l'ordre d'affectation des variables dans (CP)RBBA. La performance de (CP)RBBA dépend donc de l'ordre de points. Nous comparons table 1 les performances de CPRBBA (modèle local) avec 4 ordres différents : l'ordre initial des points (init), la moyenne de 5 ordres aléatoires (rand), l'ordre de séparation des plus proches voisins comme dans RBBA (NNS) et l'ordre furthest-point first (FPF). On peut constater que l'ordre donnant les meilleurs résultats diffère d'une base à une autre. Dans ce qui suit, l'ordre FPF est utilisé car il obtient en moyenne le meilleur temps.

dataset	N	K	init	random	NNS	FPF
ruspini	75	4	0.06	**0.00**	0.01	0.01
soybean	47	4	773.91	10.01	**0.80**	1.28
hatco	100	2	0.19	**0.02**	0.07	0.05
hatco	100	3	4.68	0.69	0.55	**0.20**
hatco	100	4	980.35	556.33	78.37	**7.52**
hatco	100	5	1800+	1800+	1800+	**1636.41**
iris	150	3	1800+	**0.95**	2.30	1.33
wine	178	3	1800+	1800+	**16.37**	53.57
seeds	210	3	1800+	491.03	1353.26	**170.67**
breast	569	2	**1167.62**	1800+	1800+	1800+
moyenne			1012.7	645.9	505.2	367.1

TABLE 1 – Temps de calcul en secondes de CPRBBA avec différents ordres de points.

1. http://www.psiheart.net/QuantPsych/monograph.html
2. http://www.cp4clustering.com/
3. https://dtai.cs.kuleuven.be/CP4IM/cccg/
4. http://archive.ics.uci.edu/ml/

Comparaisons dans le cas sans contraintes. Nous comparons CPRBBA à RBBA implanté par Brusco [6], à CPClustering fondé sur la PPC [9] et à CCCG qui utilise la génération de colonnes [4]. D'autre méthodes exactes n'ont pas de codes accessibles publiquement, mais les expérimentations respectives indiquent que RBBA est le meilleur algorithm pour des valeurs K faibles, ce qui est en général le cas en Data Mining.

Les résultats sont montrés dans la table 2. Notons que RBBA et CPRBBA sont toujours meilleurs que CPClus et CCCG, et que CPRBBA même étant un cadre générique est compétitif par rapport à RBBA qui est un algorithme dédié.

	K	CCCG	CPClus	RBBA	CPRBBA
ruspini	4	1800+	0.41	**0.01**	**0.01**
soybean	4	1800+	1.21	**0.38**	1.32
hatco	2	1800+	1.74	**0.03**	0.05
hatco	3	1800+	186.18	0.29	**0.20**
hatco	4	1800+	1800+	53.95	**7.88**
hatco	5	1800+	1800+	1800+	**1636.95**
iris	3	1800+	583.19	**1.14**	1.35
wine	3	1800+	1800+	**7.86**	53.60
seeds	3	1800+	1800+	542.74	**170.67**
breast	2	1800+	1800+	1800+	1800+

TABLE 2 – Temps de calcul en secondes des approches exactes

Contraintes must-link et cannot-link. Les contraintes must-link (ML) et cannot-link (CL) sont générées aléatoirement à partir des classes réelles. Pour cela, deux points sont choisis aléatoirement, et, en fonction de leur classe une contrainte ML ou CL est créée, jusqu'à ce que le nombre souhaité de contraintes soit atteint.

Contraintes ML. On constate (Table 3) que CPRBBA est meilleur que deux approches exactes actuelles pour le clustering sous contraintes CCCG et CPClus. Avec des contraintes ML, on ne note pas de grandes différences entre les modèles local et complet, car on utilise les ML-blocs au lieu des points individuels initiaux. Pour un seul cas de wine avec 50 contraintes, CPRBBA ne peut pas terminer la recherche avant la limite de temps imparti.

Contraintes CL. Les résultats avec des contraintes cannot-link sont présentés Table 4. Ajouter des contraintes CL peut rendre le problème plus difficile. On constate encore que CPRBBA est meilleur que les autres approches, ce qui peut s'expliquer par le fait qu'il est meilleur dans le cas sans contraintes. Plus on ajoute des contraintes CL, plus il existe de cas où la solution optimale n'est pas prouvée dans la limite de temps (voir nombre entre parenthèses), ce qui implique

	#c	cccg	cpclus	cprbba-local	cprbba-full
iris	10	1800+ (5)	341.59 (0)	**0.81 (0)**	0.86 (0)
iris	25	1800+ (5)	288.58 (0)	**1.51 (0)**	1.58 (0)
iris	50	1800+ (5)	135.32 (0)	**0.23 (0)**	0.25 (0)
iris	100	47.20 (0)	1.20 (0)	**0.01 (0)**	**0.01 (0)**
iris	150	0.20 (0)	0.07 (0)	**0.01 (0)**	**0.01 (0)**
wine	10	1800+ (5)	1800+ (5)	**258.54 (0)**	259.30 (0)
wine	25	1800+ (5)	1800+ (5)	**217.29 (0)**	218.76 (0)
wine	50	1800+ (5)	1800+ (5)	**363.34 (1)**	363.62 (1)
wine	100	1800+ (5)	1800+ (5)	**1.19 (0)**	1.23 (0)
wine	150	10.60 (0)	18.92 (0)	**0.13 (0)**	**0.13 (0)**

TABLE 3 – Temps moyen sur 5 jeux aléatoires de #c contraintes ML ; entre parenthèses le nombre de cas où le temps limit de 1800 secondes est dépassé.

un temps élevé en moyenne. On peut observer l'intérêt du modèle complet avec la base iris et en considérant plus de 100 contraintes CL.

	#c	cccg	cpclus	cprbba-local	cprbba-full
iris	10	1800+ (5)	727.32 (0)	**1.69 (0)**	1.79 (0)
iris	25	1800+ (5)	685.50 (0)	**6.38 (0)**	6.70 (0)
iris	50	1800+ (5)	1694.03 (4)	**63.94 (0)**	64.07 (0)
iris	100	1800+ (5)	497.90 (0)	368.41 (1)	**15.40 (0)**
iris	150	1800+ (5)	643.72 (1)	721.29 (2)	**361.57 (1)**
iris	250	1800+ (5)	1094.49 (3)	1080.66 (3)	**0.74 (0)**
wine	10	1800+ (5)	1800+ (5)	**622.89 (1)**	625.64 (1)
wine	25	1800+ (5)	1800+ (5)	**1310.99 (2)**	1326.51 (2)
wine	50	1800+ (5)	1800+ (5)	**1697.94 (4)**	1706.36 (4)
wine	100	1800+ (5)	1800+ (5)	1800+ (5)	1800+ (5)

TABLE 4 – Temps moyen sur 5 jeux aléatoires de #c contraintes CL ; entre parenthèses nombre de cas dépassant le temps limite de 1800 secondes.

Contraintes ML et CL. Les résultats précédents se retrouvent lorsque l'on considère des combinaisons de contraintes ML et CL (Table 5). Notons que CPClus est meilleur en moyenne sur un cas (wine, 100 ML et 100 CL), car CPRBBA ne termine pas avant le temps imparti pour un jeu de contraintes (2 pour le modèle local).

	#c	cccg	cpclus	cprbba-local	cprbba-full
iris	10	1800+ (5)	468.48 (0)	**1.13 (0)**	1.22 (0)
iris	25	1800+ (5)	204.89 (0)	**1.41 (0)**	1.56 (0)
iris	50	1800+ (5)	205.98 (0)	**0.27 (0)**	0.34 (0)
iris	100	279.80 (0)	0.09 (0)	**0.01 (0)**	**0.01 (0)**
iris	150	0.20 (0)	0.02 (0)	0.02 (0)	**0.01 (0)**
wine	10	1800+ (5)	1800+ (5)	**1029.67 (2)**	1033.67 (2)
wine	25	1800+ (5)	1800+ (5)	**724.33 (2)**	724.68 (2)
wine	50	1800+ (5)	1800+ (5)	749.87 (2)	**749.46 (2)**
wine	100	1800+ (5)	**21.58 (0)**	1132.24 (2)	361.90 (1)
wine	150	172.80 (0)	0.08 (0)	3.83 (0)	**0.04 (0)**
wine	250	0.20 (0)	0.02 (0)	**0.01 (0)**	**0.01 (0)**

TABLE 5 – Temps moyen sur 5 jeux aléatoires de #c contraintes ML et #c contraintes CL ; entre parenthèses nombre de cas dépassant le temps limite de 1800 secondes.

Contraintes de taille de clusters. Pour une contrainte de taille minimale, CPRBBA est meilleur que CPClus (Table 6). On observe des résultats similaires pour une contrainte de taille maximale (Table 7).

	K	taille	cpclus	cprbba-local	cprbba-full
ruspini	4	16	0.47	**0.00**	0.26
ruspini	4	17	1.08	**0.02**	1.17
ruspini	4	18	270.00	**9.00**	24.06
soybean	4	10	1.28	**1.39**	1.78
soybean	4	11	1800+	**1563.12**	1652.13
iris	3	38	564.86	**1.32**	1.67
iris	3	42	693.38	9.23	**2.45**
iris	3	46	933.23	341.23	**18.46**
iris	3	50	1508.77	1800+	**294.75**

TABLE 6 – Contrainte de taille minimale des clusters

	K	taille	cpclus	cprbba-local	cprbba-full
ruspini	4	21	0.46	**0.01**	0.02
ruspini	4	20	0.54	**0.01**	0.05
ruspini	4	19	1800+	**602.82**	794.83
soybean	4	14	**1.28**	1.32	1.83
soybean	4	13	17.52	**13.19**	17.44
iris	3	62	589.92	**1.31**	1.67
iris	3	58	723.63	3.95	**3.04**
iris	3	54	973.09	96.78	**18.31**
iris	3	50	1483.88	1800+	**158.75**

TABLE 7 – Contrainte de taille maximale des clusters

Nous observons que, dans tous les cas avec contraintes utilisateur, CPRBBA obtient les meilleurs résultats, comparé aux autres méthodes exactes. Dans le cas sans contrainte utilisateur, CPRBBA est compétitif par rapport à l'algorithme dédié RBBA.

7 Conclusion

Dans ce papier, nous étudions la tâche de clustering sous contraintes minimisant la somme des carrés en présence de contraintes utilisateur. Nous considérons l'algorithme *branch-and-bound* répétitif, l'une des meilleures méthodes exactes pour le cas sans contraintes utilisateur. Nous avons proposé une extension de cet algorithme pour intégrer des contraintes utilisateur et nous avons réalisé un cadre général à base de programmation par contraintes (PPC) pour cette extension. La PPC est utilisée pour des étapes *branch-and-bound* et aussi pour le calcul de bornes supérieures et inférieures. Nous proposons deux modèles en PPC pour trouver de meilleures bornes inférieures, ainsi qu' un mécanisme de propagation spécifique pour mieux exploiter les bornes trouvées. Les expérimentations sur des bases de données classiques montrent que notre méthode, bien qu'elle soit générique, est compétitive par rapport à l'algorithme dédié RBBA dans le

cas sans contraintes. Pour les cas avec contraintes utilisateur, notre méthode est toujours meilleure que les méthodes exactes existantes.

Pour améliorer davantage la performance de notre cadre, une direction prometteuse est l'étude des heuristiques d'ordonnancement des points, voire même l'étude d'un ordre dynamique, ainsi que le choix des valeurs pendant le *branch-and-bound*. De plus, puisque RBBA a aussi été appliqué à des tâches de clustering optimisant d'autres critères, par exemple la somme des dissimilarités intra-cluster, notre approche peut être étendue pour ces tâches de clustering en présence de contraintes utilisateur.

Références

[1] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean Sum-of-squares Clustering. *Machine Learning*, 75(2) :245–248, 2009.

[2] Daniel Aloise and Pierre Hansen. An branch-and-cut SDP-based algorithm for minimum sum-of-squares clustering. *Pesquisa Operacional*, 29(3) :503–516, 2009.

[3] Daniel Aloise, Pierre Hansen, and Leo Liberti. An improved column generation algorithm for minimum sum-of-squares clustering. *Mathematical Programming*, 131(1-2) :195–220, 2012.

[4] Behrouz Babaki, Tias Guns, and Siegfried Nijssen. Constrained clustering using column generation. In *Proc. CPAIOR*, pages 438–454, 2014.

[5] Michael J. Brusco. An enhanced branch-and-bound algorithm for a partitioning problem. *British Journal of Mathematical and Statistical Psychology*, 56(1) :83–92, 2003.

[6] Michael J. Brusco. A repetitive branch-and-bound procedure for minimum within-cluster sums of squares partitioning. *Psychometrika*, 71(2) :347–363, 2006.

[7] Thi-Bich-Hanh Dao, Kanh-Chuong Duong, and Christel Vrain. A Declarative Framework for Constrained Clustering. In *Proc. ECML/PKDD*, pages 419–434, 2013.

[8] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. *Artificial Intelligence*, page DOI : 10.1016/j.artint.2015.05.006, 2015.

[9] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained minimum sum of squares clustering by constraint programming. In *CP, Proceedings*, pages 557–573, 2015.

[10] Ian Davidson, S. S. Ravi, and Leonid Shamis. A SAT-based Framework for Efficient Constrained Clustering. In *Proc. ICDM*, pages 94–105, 2010.

[11] O. du Merle, P. Hansen, B. Jaumard, and N. Mladenovic. An interior point algorithm for minimum sum-of-squares clustering. *SIAM Journal on Scientific Computing*, 21(4) :1485–1505, 1999.

[12] A. W. F. Edwards and L. L. Cavalli-Sforza. A method for cluster analysis. *Biometrics*, 21(2) :362–375, 1965.

[13] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38 :293–306, 1985.

[14] Pierre Hansen and Brigitte Jaumard. Cluster analysis and mathematical programming. *Mathematical Programming*, 79(1-3) :191–215, 1997.

[15] Robert E. Jensen. A dynamic programming algorithm for cluster analysis. *Journal of the Operations Research Society of America*, 7 :1034–1057, 1969.

[16] W. L. G. Koontz, P. M. Narendra, and K. Fukunaga. A branch and bound clustering algorithm. *IEEE Trans. Comput.*, 24(9) :908–915, 1975.

[17] Marianne Mueller and Stefan Kramer. Integer Linear Programming Models for Constrained Clustering. In *Proc. DS*, pages 159–173, 2010.

[18] Dan Pelleg and Dorit Baras. K-means with large and noisy constraint sets. In *ECML*, volume 4701, pages 674–682, 2007.

[19] Douglas Steinley. k-means clustering : A half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59(1) :1–34, 2006.

[20] B.J. van Os and J.J. Meulman. Improving Dynamic Programming Strategies for Partitioning. *Journal of Classification*, 2004.

[21] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proc. ICML*, pages 1103–1110, 2000.

[22] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained K-means Clustering with Background Knowledge. In *Proc. ICML*, pages 577–584, 2001.

[23] Y. Xia and J. Peng. A cutting algorithm for the minimum sum-of-squared error clustering. In *SDM*, 2005.

Un algorithme pour une extension du problème de plus court chemin contraint. Application à l'optimisation des routes maritimes.

Estelle Chauveau[1,2] **Philippe Jegou**[2] **Nicolas Prcovic**[2]

[1] Atos, 83140 Six-Fours-les-Plages
[2] Aix-Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296, 13397 Marseille Cedex 20
{estelle.chauveau, philippe.jegou, nicolas.prcovic}@lsis.org

Résumé

Cet article traite du problème d'optimisation des routes maritimes en fonction des conditions météorologiques en temps réel. Ce problème relève des questions de calcul de plus court chemin multiobjectif ainsi que des problèmes de plus court chemin dans un graphe à valuations dynamiques. Les questions de calculs de plus courts chemins multicritères ont fait l'objet de nombreux travaux de même que pour le cas monocritère avec des valuations dynamiques. La problématique associant ces deux difficultés rend la question significativement plus complexe et les résultats obtenus jusqu'à présent sont largement moins satisfaisants. Dans cet article, nous proposons une extension au cas des graphes à valuations dynamiques de l'algorithme NAMOA* qui traite du calcul de plus court chemin multiobjectif mais dans un cadre seulement statique. Nous étudions ensuite les propriétés de base du nouvel algorithme.

Abstract

The article deals with the weather routing problem for cargo ships, which involves optimizing the ship's route with real-time weather informations. This problem can be modelled with a multiobjective shortest path problem in a time-dependent graph. Multicriteria shortest path problems are widely studied in the literature. Monocriteria shortest path in time-dependent graphs as well has been deeply studied too, since it has numerous application, especially in transportation field. However, the combination of both issues is not studied as much as each one separatly, although it is a very complex problem. In this paper, we propose an extension of the algorithm NAMOA* to the case of directed graphs with dynamic valuations since this algorithm deals with the calculation of shortest path multiobjectif but only in a static framework. We study the basic properties of the new algorithm.

1 Introduction

Le transport de marchandises par voies maritimes domine le marché puisqu'il représente 80% du commerce mondial en termes de volume [1]. Pour ce type de transport, 40% des frais opérationnels sont absorbés par le prix du carburant lors du voyage [16]. La diminution de la vitesse du navire entraîne une baisse de la consommation en carburant, et l'optimisation de ces frais de carburant entre parfois en conflit avec l'objectif de minimiser la durée du trajet. Par ailleurs, il peut parfois être nécessaire de contourner des zones dangereuses, à cause des risques de piraterie, ou des risques météorologiques : ce détour implique une augmentation de la durée totale de transport et de la consommation de carburant. Ces objectifs sont contradictoires. Ils dépendent de paramètres météorologiques qui changent dans le temps, introduisant de fait un caractère dynamique à cette problématique. Les armateurs sont amenés à s'appuyer sur des outils d'aide à la décision sophistiqués pour traiter la question du choix du chemin de meilleur compromis. Toutefois, les outils existants prennent rarement en compte de manière satisfaisante l'ensemble des questions soulevées, et au-delà, il s'agit d'un problème complexe.

À partir du problème de décision connu sous le vocable de Shortest Weight-Constrained Path, qui est NP-complet [10], on peut facilement démontrer que le problème de plus court chemin bicritère est NP-difficile (problème d'optimisation associé). Naturellement, l'augmentation du nombre de critères

accroît la combinatoire du problème. À cela vient se greffer une difficulté additionnelle, l'aspect dynamique de la pondération des arcs dans la mesure où celle-ci peut évoluer en fonction du temps, comme l'impose la possible évolution des conditions météorologiques pendant la durée d'un trajet.

Cela étant, il existe une littérature non négligeable qui traite de ces questions en envisageant la problématique de plus court chemin multiobjectif d'une part, celle de recherches de chemins monocritères à valuations dynamiques d'autre part, et enfin ces deux problématiques simultanément. Nous nous positionnons dans ce dernier cas. Parmi les algorithmes de plus court chemin multiobjectif, les algorithmes à fixation d'étiquette se révèlent particulièrement efficaces en pratique. Au sein de ce dernier ensemble, NAMOA* (introduit par Mandow et De la Cruz dans [21]) se différencie par l'utilisation d'une heuristique dotée de propriétés - nous les développons plus loin - lui assurant de fournir l'ensemble des chemins optimaux en visitant un minimum de nœuds. Le graphe de recherche dans lequel nous allons travailler, qui correspond en fait à un maillage de l'océan, bénéficiera fortement de ce cadre heuristique qui permet d'ordonner l'exploration des chemins. Enfin, cette heuristique permet de filtrer certaines solutions partielles qui n'aboutiront pas à une solution optimale. Pour cette raison, nous avons décidé d'étendre l'algorithme NAMOA* au cas d'un graphe à valuations dynamiques pour résoudre notre problème. Cette extension se fait assez naturellement en intégrant le temps aux critères d'optimisation. Le nouvel algorithme que nous proposerons sera appelé NAMOA*-T.

Dans cette contribution, nous présentons dans un premier temps (en section 2) une modélisation mathématique du problème dans le cas de l'optimisation des routes maritimes. Cette modélisation laissera place à un état de l'art (section 3) récapitulant les différents travaux susceptibles de nous intéresser. Certaines caractéristiques ou propriétés de notre problème sont mises en évidence dans la section 4 : elles permettront d'expliquer le choix d'extension de NAMOA*. Ce dernier algorithme est présenté en détail (section 5), avant de présenter notre algorithme NAMOA*-T (section 6). Avant de conclure, une propriété essentielle de NAMOA*-T est mise en évidence.

2 Formalisation du problème

2.1 Problèmes de plus court chemin classiques

Le problème de plus court chemin dans un graphe est simple, et se résout rapidement avec des algorithmes tel que celui de Dijkstra pour le cas de valuations positives, ce qui sera vérifié dans notre contexte. Il est cependant connu que dans sa version multiobjective (multiobjective shortest path problem - MSPP - dont la première formulation a été introduite par Vincke dans [29]), ce problème est NP-difficile ([14]). Dans cette version, il n'existe pas une unique solution, mais un ensemble de solutions que l'on appelle *chemins pareto-optimaux* (cette notion est définie par la suite). Dans [14], Hansen met en évidence une classe de graphes pour lesquels le nombre de solutions, ie. de chemins pareto-optimaux, est exponentiel en fonction du nombre de noeuds du graphe. Notons qu'il existe une sous-classe de problèmes MSPP pour lesquels la fonction de coût des arcs possède des caractéristiques-qui rendent le problème polynomial. Par exemple, dans le cas biobjectif, si l'un des coûts est une fonction linéaire de l'autre, il est trivial que cela peut être ramené à un problème monocritère. En ce qui concerne le problème que nous étudions, les fonctions de coûts (durée de voyage, consommation de carburant) au regard de la météo sont des fonctions complexes qui ne sont pas dotées de ce type de propriétés.
Le problème que nous étudions correspond au MSPP dans un graphe dans lequel le coût de chacun des arcs dépend du temps, et de ce fait, les algorithmes proposés pour résoudre le MSPP ne suffisent plus. Dans la partie suivante, nous présentons le problème de manière formelle, et nous posons les notations qui seront utilisées par la suite.

2.2 Plus court chemin multicritère dans un graphe à valuations dynamiques

Nous proposons de modéliser le problème de déplacement sur un océan dans les termes d'un problème de plus court chemin multicritère dans un graphe à valuations dynamiques où le temps est discrétisé. Par "graphe à valuations dynamiques", nous entendons un graphe dans lequel la pondération de chaque arc est une fonction du temps. Dans ce graphe, lorsqu'un arc existe d'un sommet i vers un sommet j, alors l'arc réciproque existe nécessairement et son coût ne sera pas nécessairement identique. Il s'agit donc d'un graphe avec circuits dans lequel les valuations ont des valeurs positives (temps, consommation de carburant, risques divers...). Formellement, il s'agit d'un graphe orienté muni d'une fonction de coût $\vec{c}$, soit un objet $G = (N, A, T, \vec{c})$. Pour définir le problème, nous distinguons deux nœuds $o, d \in N$ (respectivement l'origine et la destination) ainsi qu'une date $t_0 \in T$. N est un ensemble fini de nœuds $\{i_1, i_2, \dots i_n\}$, A un ensemble d'arcs de la forme (i, j) et T une séquence finie de dates $< t_0, t_1, t_2, \dots >$. $\vec{c}_{ij}(t)$ est le coût de la traversée d'un arc (i, j) en quittant le sommet i à la date $t \in T$; c'est un vecteur de dimension k, ou k est le nombre de critères du problème. On notera

que toutes les valuations sont dépendantes du temps. Par convention, le premier élément $[\vec{c}_{ij}(t)]_1$ du vecteur de coût est la durée de traversée de l'arc (i, j). Un chemin (nous ne considèrerons que les chemins élémentaires ici) dans G de i_0 vers i_q est une séquence de q arcs $< (i_0, i_1), (i_1, i_2), \ldots, (i_{q-1}, i_q) >$ dans laquelle $i_0, i_1, \ldots i_q$ sont des nœuds distincts. Soit $P_{ij}(t)$ l'ensemble des chemins de i vers j tels que le sommet i est quitté à la date t, et que le sommet j est atteint à une date $t' \in T$. Étant donné un chemin $p_{ij}(t) \in P_{ij}(t)$, avec $j \neq i$, on définit récursivement le coût total $\vec{C}$ de ce chemin. Si $p_{ij}(t)$ possède un unique arc $(i, j) \in A$, alors $\vec{C}(p_{ij}(t)) = \vec{c}_{ij}(t)$. Sinon, le chemin $p_{ij}(t)$ est composé du chemin p_{ih} suivi de l'arc (h, j), auquel cas, $\vec{C}(p_{ij}(t)) = \vec{C}(p_{ih}(t)) + \vec{c}_{hj}(t')$ avec $t' = t + [\vec{C}(p_{ih}(t))]_1$. La solution attendue du problème est le sous-ensemble $P^*_{od}(t_0) \subseteq P_{od}(t_0)$ de chemins non dominés (au sens de Pareto), avec $p^*_{od}(t_0) \in P^*_{od}(t_0)$ si et seulement si :

$$\nexists p_{od}(t_0) \in P_{od}(t_0) \mid \vec{C}(p_{od}(t_0)) \prec \vec{C}(p^*_{od}(t_0))$$

où $\prec$ est la relation d'ordre partiel appelée *dominance* [20] dont le définition est rappelée dans la section 2.3.

Dans le cadre de nos recherches, il est important d'étudier les algorithmes fournissant l'ensemble des vecteurs pareto-optimaux. Il s'agira alors de développer les algorithmes retenus et de les intégrer à un logiciel (en cours de développement chez Atos), puis de les tester sur nos jeux de données. Ce seront alors des algorithmes de référence, fournissant les solutions de référence. Il sera par la suite intéressant d'évaluer si le temps de calcul permet l'utilisation de ces algorithmes en temps réel. Dans le cas contraire, il conviendra d'étudier des algorithmes plus rapides, fournissant un sous-ensemble des solutions pareto-optimales. Ainsi, il sera possible de comparer les deux types d'algorithmes (qualité des solutions fournies et temps de calcul).

2.3 Définitions

Dans cette section, quelques définitions nécessaires par la suite sont rappelées.

Définition 1 (Graphe FIFO)**:**
Un graphe $G = (N, A, T, \vec{c})$ est dit FIFO si $\forall (i, j) \in A$, $\forall t, t' \in T$ avec $t < t'$, on a :

$$t + [\vec{c}_{ij}(t)]_1 \leq t' + [\vec{c}_{ij}(t')]_1$$

En d'autres termes, un graphe est FIFO si partir d'un nœud i à une date t' postérieure à t ne permettra jamais d'arriver plus tôt à un nœud j.

Définition 2 (Dominance de Pareto)**:**
Dans un contexte de minimisation, un vecteur coût $\vec{c}$ de dimension k domine un vecteur coût $\vec{c}'$ de dimension k si :

$$\forall i \in [1 \ldots k],\ [\vec{c}\,]_i \leq [\vec{c}']_i \text{ et } \vec{c} \neq \vec{c}' \qquad (1)$$

Cela se note $\vec{c} \prec \vec{c}'$.

On remarque que la relation de dominance est bien entendu transitive.

Définition 3 (Dominance de Pareto (2))**:**
Dans un contexte de minimisation, un ensemble C de vecteurs coûts de dimension k domine un vecteur coût $\vec{c}'$ de dimension k si :

$$\exists \vec{c} \in C \quad t.q. \quad \vec{c} \prec \vec{c}' \qquad (2)$$

Cela se note $C \prec \vec{c}$.

Définition 4 (Pareto-optimalité)**:**
Un vecteur $\vec{c}$ appartenant à un ensemble C de vecteurs est pareto-optimal au sein de cet ensemble si et seulement si $C \not\prec \vec{c}$.

Définition 5 (Principe d'optimalité)**:**
Le principe d'optimalité dans un graphe assure que les sous-chemins d'un chemin optimal sont des chemins optimaux (et ce quelque soit le critère d'optimalité choisi).

3 Travaux antérieurs

La complexité de l'optimisation des routes maritimes provient de ses deux principales caractéristiques : elle est multicritère et dépendante du temps. Les applications faisant appel à la combinaison de ces deux propriétés sont nombreuses, notamment dans le domaine des transports (routier, ferroviaire, etc.). Elles ont conduit à la production de nombreux algorithmes que nous évoquons dans cette section qui est organisée en présentant dans l'ordre les travaux traitant :

- des problèmes de plus court chemin multiobjectif ;
- des problèmes de cheminement dans un graphe monocritère à valuations dynamiques ;
- de ces deux problématiques simultanément.

Par ailleurs, sont présentés ici exclusivement les algorithmes admissibles (c'est-à-dire que si une solution existe, l'algorithme garantit de trouver la (resp. les) solution(s) optimale(s)) avec comme critère d'optimalité la pareto-optimalité.

L'étude bibliographique qui suit montre que les algorithmes de plus court chemin multiobjectif appartiennent généralement à l'une des catégories suivantes : les algorithmes dits à *étiquettes*, les *méthodes de classe-*

ment, ou les approches *paramétriques*. Geoffrion étudie en 1967 la résolution de programmes mathématiques bicritères de manière générale [11], et c'est Hansen qui se penche en premier sur la question du plus court chemin dans un graphe bicritère. Hansen [14] présente dix problèmes bicritères différents. Il mentionne à cette occasion un algorithme à étiquettes, c'est-à-dire qu'il utilise un système d'étiquettes associées à chaque sommet du graphe indiquant à tout moment des informations concernant le (les) plus court(s) chemin(s) identifié(s) du sommet origine jusqu'au sommet courant. Climaco et Martins [5] proposent un algorithme de classement (classement des k plus courts chemins) dont l'efficacité empirique semble moindre comparée aux autres méthodes. En 1984 Martins développe une version pluri-étiquettes [22] de l'algorithme de Dijkstra afin d'identifier toutes les solutions pareto-optimales du sommet origine vers tous les sommets du graphe. L'année suivante, Corley et Moon utilisent la programmation dynamique pour résoudre le problème de plus court chemin multicritère [7]. Les algorithmes à étiquettes sont les plus étudiés puisqu'ils fournissent en général de bons résultats, et différentes évolutions ont étés proposées au fil du temps dont [15], [9], [2], et plus récemment en 2005, [21] qui se démarque par l'utilisation d'une heuristique. Ce dernier article présente l'algorithme NAMOA* qui sera détaillé en section 5. Dans cet article, les auteurs prouvent que si l'heuristique est *consistante* (c'est-à-dire monotone), alors l'algorithme est admissible. Mote, quant à lui, aborde le problème en se basant sur une approche paramétrique en deux étapes [23]. Dans la première étape il relâche le problème en effectuant une combinaison convexe sur les deux objectifs à minimiser, résout ce problème relâché par une méthode classique d'optimisation linéaire, et identifie un certain nombre de chemins pareto-optimaux. Dans la seconde étape de l'algorithme, il retrouve les chemins pareto-optimaux manquants grâce à une procédure à correction d'étiquettes. De la même manière et plus récemment, Sedeño-Noda et Raith [26] proposent une méthode permettant d'identifier les solutions non dominées "supportées" (ce qui correspond à la première étape de l'algorithme de Mote), et ce pour un problème biobjectif. Dans les travaux plus appliqués on peut citer la thèse de Sauvanet [25] qui se penche sur la problématique de choix d'itinéraire pour les cyclistes ; il distingue les approches *a posteriori* (détermination de l'ensemble du front de pareto) des approches *a priori* (solutions de meilleur compromis). L'étude des approches *a priori* dans le cadre du maritime pourra faire l'objet d'un prochain travail.

Enfin, pour une revue plus approfondie de la littérature dans le domaine des plus courts trajets multicritères, l'étude d'Ehrgott et Gandibleux [20] est une référence.

Le problème monocritère de plus court chemin dans un graphe à valuations dynamiques trouve ses racines dans l'article de Cooke et Halsey [6] de 1969, dans lequel les temps de trajet sur un arc sont considérés comme des fonctions générales du temps, et une grille de valeurs discrètes du temps est utilisée. Ils décrivent un algorithme basé sur le principe d'optimalité de Bellman [3]. Le problème est ensuite repris par Dreyfus en 1969 qui propose une méthode basée sur l'algorithme de Dijkstra [8]. Il explique que le problème peut être traité de manière aussi efficace que son homologue statique. Cependant Kaufman et Smith prouvent en 1993 que cette assertion n'est vraie que dans le cas d'un graphe FIFO [18]. La même année, Ziliaskopoulos et Mahmassani suggèrent un algorithme à correction d'étiquettes résolvant le problème en question dans un graphe ne possédant pas la propriété FIFO. Orda et Rom étudient dans [24] les différentes versions du problème (avec ou sans la possibilité d'attendre à un sommet, avec ou sans date de départ, temps à valeurs discrètes ou continues), et fournissent des algorithmes pour ces différents cas. Chabini [4] présente en 1998 un algorithme à fixation d'étiquettes identifiant les meilleurs chemins pour toutes les dates de départ du sommet origine, cela dans un graphe non FIFO. Enfin, Karoulas mentionne en 2006 une version de A* répondant au problème [17]. De toute évidence, le problème que l'on étudie (dans un graphe FIFO avec des valeurs de temps discrètes) est généralement traité efficacement en se basant sur des algorithmes statiques à fixation ou correction d'étiquettes.

La combinaison des deux problématiques (plus court chemin multiobjectif dans un graphe à valuations dynamiques) a été envisagée pour la première fois dans les années 90, notamment par Kostreva et Wiecek [19] qui s'appuient sur les travaux de Cooke et Halsey [6]. Ils proposent deux algorithmes basés sur les principes de la programmation dynamique. Leurs travaux portent sur un graphe FIFO dans lequel les fonctions de coût des arcs (par rapport au temps) sont non décroissantes. En 2000, Getachew et al. [27] adaptent les travaux de Kostreva et Wiecek à un graphe non FIFO comportant des fonctions de coût des arcs non nécessairement croissantes, mais bornées (cf. borne supérieure et borne inférieure). Les travaux de Hamacher et al. (2006) présentés dans [13] montrent que dans un graphe FIFO dans lequel l'attente n'est pas possible au niveau d'un sommet, le principe d'optimalité de Bellman n'est vrai qu'en *arrière*, c'est-à-dire dans la cas où l'algorithme visite en premier le sommet destination, puis ses prédécesseurs et ainsi de suite. Ils proposent ensuite un algorithme à fixation d'étiquettes et le comparent avec l'algorithme de Kostreva et Wiecek. Ils montrent ainsi la supériorité de leur algorithme à étiquettes par rapport à l'algorithme basé sur le prin-

cipe d'optimalité de Bellman.

Pour conclure cet état de l'art, on peut citer les travaux appliqués à des problèmes pratiques : Veneti et al. [28] qui se sont penchés sur le problème de plus court chemin dans un graphe FIFO, et ce dans le contexte de l'optimisation des routes maritimes. Ils proposent un algorithme à fixation d'étiquette basé sur celui de martins et comparent leurs résultats à ceux de Hamacher. Enfin, la thèse de Gräbener et al. [12] s'inscrit dans la thématique du calcul d'itinéraire en milieu urbain. Il balaye les problématiques communes à celle de l'optimisation des routes maritimes, à savoir la notion de multiobjectif et de dépendance temporelle. Il étend notamment un algorithme de Martins, en vue d'identifier l'ensemble des solutions pareto-optimales, mais propose aussi des approches permettant la sélection de sous-ensembles de qualité.

4 Propriétés du problème

4.1 Propriétés

L'ensemble des modèles étudiés dans l'état de l'art se subdivise en deux parties : les modèles à temps discret et les modèles à temps continu. Les données météorologiques en amont de notre modèle sont des données discrètes. À titre d'exemple, il existe des fichiers de prévisions météorologiques contenant des informations sur les vents, les courants (etc.) toutes les 6 heures. Afin d'être cohérent avec les sources de prévisions météorologiques, le modèle mathématique utilisé sera un **modèle à temps discrétisé**.

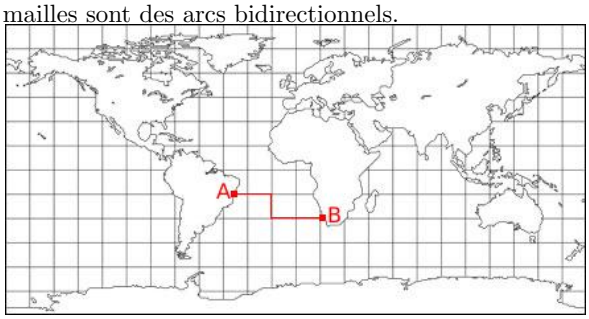
Par ailleurs, les navires de transport de marchandise sont dotés de moteurs qui supportent mal les basses vitesses. Le bateau ne pouvant aller en dessous d'une certaine vitesse, il n'est pas nécessaire de prendre en compte, dans notre modèle, la possibilité d'attendre au niveau d'un sommet (par exemple pour que les conditions météorologiques s'améliorent).

Dans la modélisation que nous proposons, les nœuds représentent des lieux géographiques et les arcs des déplacements possibles entre ces nœuds. Les différents déplacements autorisés doivent être définis au préalable : un exemple illustratif est donné en fig. 1. Ainsi, nous aurons affaire à un graphe dont les arcs sont bidirectionnels, et pour lequel le coût d'un arc n'est pas nécessairement égal à l'arc réciproque. Cela implique que le graphe de travail est un graphe comportant des circuits.

Il est important de noter que nous ne traiterons que des **valuations positives** (temps, consommation de carburant, risques divers...) sur les arcs du graphe, ce qui limite la difficulté du problème du fait de l'absence de circuits absorbants (circuits de coût négatif).

Le plus court chemin dans un graphe à valuations dynamiques nécessite l'étude de certaines propriétés du problème, notamment la propriété dite *FIFO* (Définition 1). Dans le cadre de l'optimisation des routes maritimes, **la donnée d'entrée est un graphe FIFO**. Enfin, le principe d'optimalité sur lequel se fondent de nombreux algorithmes est remis en cause (cf. Définition 5) :

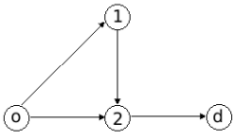
FIG. 1 – Exemple de déplacements possibles modélisables par un graphe "maillage". Chaque intersection du maillage correspond à un nœud, et les contours des mailles sont des arcs bidirectionnels.



Théorème 1 (Non conservation du principe d'optimalité dans un graphe dynamique)**.** *Dans un graphe à valuations dynamiques, le principe d'optimalité n'est pas conservé : un chemin pareto-optimal n'est pas nécessairement constitué de sous-chemins pareto-optimaux.*

On propose un contre-exemple similaire à celui de Hamacher dans [13] permettant de démontrer ce théorème :

FIG. 2 – Graphe exemple



Preuve 1. *On considère le graphe bicritère* $G = (N, A, T, \vec{c})$ *illustré en figure 2. Les données de ce graphe sont les suivantes :*
$\vec{c}_{o1}(t = 0) = (1, 1)$
$\vec{c}_{o2}(t = 0) = (1, 1)$
$\vec{c}_{12}(t = 1) = (1, 1)$
$\vec{c}_{2d}(t = 1) = (2, 2)$
$\vec{c}_{2d}(t = 2) = (1, 1)$
On rappelle que dans notre notation, le premier élément du vecteur de coût correspond à la durée du trajet. On quitte le sommet origine au temps $t = 0$.

Deux chemin pareto-optimaux de même coût $(3, 3)$ *sont solutions de ce problème :*
$o \to 2 \to d$ *et* $o \to 1 \to 2 \to d$.

On voit bien que ce dernier chemin est consitué du sous-chemin $o \to 1 \to 2$ *(de coût* $(2,2)$*) qui est dominé par* $o \to 2$ *(de coût* $(1,1)$*) et qui n'est donc pas pareto-optimal.* □

En pratique, la non conservation du principe d'optimalité peut se traduire de la manière suivante : il peut être intéressant de mettre plus de temps à naviguer jusqu'à un certain point en vue de laisser passer une intempérie, et de bénéficier par la suite de conditions météorologiques trés favorables.

4.2 Choix d'extension de NAMOA*

Lors d'un calcul de plus court chemin entre deux ports, certains nœuds n'ont pas besoin d'être visités par l'algorithme, puisque les meilleurs chemins auxquels ils peuvent appartenir sont de toutes manières dominés par des chemins intégraux déjà calculés (appartenant à $P_{od(t_0)}$). Par exemple, si l'on considère un trajet allant de Rio au Cap, les chemins partiels passant par l'océan Pacifique ou l'océan Indien peuvent être rapidement filtrés par l'algorithme, et ce, même si les conditions météorologiques en Atlantique sont très défavorables. Il existe des algorithmes admissibles permettant de prendre en compte cette observation, comme l'algorithme NAMOA* [21]. Cet algorithme, qui résout le problème de plus court chemin multiobjectif statique, s'appuie sur l'utilisation d'heuristiques *optimistes* permettant d'explorer en priorité les chemins partiels ayant un bon potentiel, et de filtrer certaines zones géographiques. Il évitera donc d'explorer un nombre significatif de chemins qui n'ont aucune chance d'appartenir à une solution. C'est pour cette raison que, dans le cadre de l'optimisation des routes maritimes, nous proposons une extension de NAMOA* au cas d'un graphe à valuations dynamiques. Il faudra par la suite comparer cette extension à un algorithme existant et efficace (par exemple l'algorithme de Hamacher [13]), afin de déterminer son adéquation avec le problème maritime.

Par ailleurs, l'opération de base de NAMOA* est la sélection d'un chemin pour exploration (contrairement à certains algorithmes pour lesquels l'opération de base est la sélection d'un noeud pour exploration), de ce fait, l'extension au cas du graphe dynamique pour lequel l'un des critère à optimiser est le temps se fait plus naturellement.

5 NAMOA* : présentation de l'algorithme et de ses propriétés

Dans cette partie, l'algorithme NAMOA* (pour New Algorithm for Multi-Objective A*) proposé par Mandow et De La Cruz dans [21] est présenté avec les notations de la section 2.2. Cet algorithme permet de trouver les chemins pareto-optimaux d'un graphe multicritère statique. Le principe de l'algorithme consiste à identifier des chemins partiels entre le sommet origine et les différents sommets du graphe. À chaque sommet i du graphe, les chemins partiels identifiés jusqu'à ce sommet sont caractérisés par leur coût partiel (coût du chemin de o vers i). Chaque chemin partiel peut être éliminé à tout moment grâce aux procédures de filtrage et d'élagage (détaillées dans la section 5.2). Les chemins partiels sont explorés dans un ordre permettant de prioriser les chemins dont l'estimation heuristique totale est pareto-optimale.

5.1 Propriétés de NAMOA*

NAMOA* prend en entrée un graphe multicritère statique, la notation utilisée pour définir le problème dans la section 2.2 sera donc réutilisée, à l'exception près que le temps ne sera pas pris en compte. Les données d'entrées du problème sont donc un graphe $G = (N, A, \vec{c})$ (au lieu de $G = (N, A, T, \vec{c})$), la fonction de coût $\vec{c}$ et les nœuds $o, d \in N$. Pour alléger l'écriture, il sera établi par la suite que l'optimisation des objectifs consiste en une minimisation. L'algorithme utilise une évaluation heuristique du coût entre un nœud i du graphe et le nœud destination d. On introduit la fonction $\vec{h_i}$ qui à chaque nœud i associe un vecteur de dimension k contenant les estimations heuristiques du nœud pour chacun des objectifs, ordonné de la même façon que le vecteur de coût. Par exemple si l'on considère le critère *durée de trajet* et le critère *consommation de carburant*, alors $[\vec{c}_{ij}]_1$ est la durée du trajet lors de la traversée de l'arc (i,j), $[\vec{c}_{ij}]_2$ est la consommation de carburant lors de la traversée de l'arc (i,j), $[\vec{h_i}]_1$ est l'estimation heuristique du temps de trajet du nœud i au nœud d, et $[\vec{h_i}]_2$ est l'estimation de la consommation de carburant du nœud i au nœud d.

La définition suivante va nous permettre de caractériser une propriété de NAMOA*. Soit $\overrightarrow{MIN}_i$ le vecteur coût composé des coûts minimums réels des chemins de i à d pour chaque critère.

Définition 6 (Heuristique optimiste)**:**
Pour tout $i \in N$, $\vec{h_i}$ *est une estimation heuristique strictement optimiste si*

$$\forall j \in [1 \dots k], [\vec{h_i}]_j < [\overrightarrow{MIN}_i]_j \tag{3}$$

On note cela $\vec{h_i} < \overrightarrow{MIN}_i$.

On remarque que : $\vec{h_i} < \overrightarrow{MIN}_i \Rightarrow \vec{h_i} \prec \overrightarrow{MIN}_i$

Propriété 1. *Lorsque l'heuristique utilisée est opti-*

miste, l'algorithme NAMOA est admissible.*

Cette propriété est démontrée dans l'article présentant l'algorithme. L'utilisation d'une heuristique optimiste pour les différents objectifs liés à l'optimisation des routes maritimes est tout à fait réalisable, ce qui permet de garantir que toutes les solutions pareto-optimales sont identifiées par l'algorithme. Lorsque l'algorithme sera étendu pour travailler sur un graphe à valuations dynamiques, il s'agira de vérifier que cette propriété est conservée. L'algorithme NAMOA* est détaillé dans la section 5.3.

5.2 Distinction entre les procédures de filtrage et d'élagage

La procédure d'*élagage* a lieu à chaque fois qu'un nouveau chemin partiel est trouvé. Son principe est le suivant. Soit un nouveau chemin partiel p_{oi} de coût $\vec{c}(p_{oi})$. S'il existe un autre chemin partiel p'_{oi} tel que p'_{oi} domine p_{oi}, alors le nouveau chemin partiel p_{oi} ne vaut pas la peine d'être exploré, on dit qu'il est *élagué*. A contrario, s'il existe un autre chemin partiel p'_{oi} tel que p_{oi} domine p'_{oi}, alors le chemin p'_{oi} ne vaut plus la peine d'être exploré ; on dit alors qu'il est *élagué*.

La procédures de *filtrage* quant à elle, s'appuie sur l'heuristique $\vec{h}$ présentée dans la section 5.1. Elle a lieu dans deux cas différents :

1. Un nouveau chemin intégral p_{od} est trouvé. Pour chaque chemin partiel p_{oi} non exploré, si l'estimation heuristique globale de p_{oi} (qui vaut le coût de p_{oi} sommé à l'évaluation heuristique $\vec{h_i}$) est dominée par le coût de p_{od}, alors le chemin partiel est supprimé ; on dit qu'il est *filtré*.
2. Un nouveau chemin partiel p_{oi} est identifié. Pour chaque chemin intégral p_{od} déjà trouvé, si p_{od} domine l'estimation heuristique globale de p_{oi} (qui vaut le coût de p_{oi} sommé à l'évaluation heuristique $\vec{h_i}$) alors le nouveau chemin partiel p_{oi} peut être supprimé ; on dit qu'il est *filtré*.

Pour synthétiser, la procédure d'élagage est une procédure locale (uniquement au niveau d'un nœud) basée sur le principe d'optimalité, tandis que la procédure de filtrage est une procédure globale (elle se fait au niveau dun nœud mais est basée sur une solution globale entre l'origine et la destination) et elle s'appuie sur l'utilisation d'une heuristique.

5.3 Algorithme NAMOA*

Dans cette section nous décrivons l'algorithme NAMOA* avec la notation introduite dans la section 2.2. NAMOA* est une extension de A* au cas multiobjectif pour laquelle l'opération de base est la sélection et l'exploration des chemins (par oppposition au nœud). Cette opération de base est complétée par les procédures de filtrage et d'élagage.

L'algorithme stocke les chemins partiels dans deux ensembles associés à chaque nœud i du graphe : $C_{open}(i)$ et $C_{closed}(i)$, qui contiennent les vecteurs coûts non dominés des chemins partiels atteignant les sommets en question. C_{open} contient les vecteurs coûts qui n'ont pas encore été explorés tandis que C_{closed} contient les vecteurs coûts qui ont déjà été explorés. Chaque vecteur coût de ces ensembles est étiqueté d'un ou plusieurs pointeurs vers ses sommets parents. L'ensemble des vecteurs de coût associé à un sommet i sera noté $\vec{C_i}$.

L'algorithme tient à jour une liste $OPEN$, contenant les chemins partiels qui peuvent être explorés plus en profondeur. Pour chaque nœud $i \in N$ et chaque vecteur de coût non dominé $\vec{c_i} \in C_{open}(i)$, une paire $(i, \vec{c_i})$ correspondante doit se trouver dans $OPEN$. Cependant, dans un souci de clarté, l'ensemble des évaluations heuristiques associées à chacune de ces paires sera incluse de manière à former un triplet $(i, \vec{c_i}, \vec{f}(i, \vec{c_i}))$, avec $\vec{f}(i, \vec{c_i}) = \vec{c_i} + \vec{h_i}$. Initialement, le triplet $(o, \vec{c_o}, \vec{f}(o, \vec{c_o}))$ est le seul triplet dans la liste $OPEN$. Dans la littérature, ce triplet est appelé "étiquette" (d'où les algorithmes dits à étiquettes).

À chaque itération, l'algorithme va considérer l'extension d'un triplet ouvert $(i, \vec{c_i}, \vec{f}(i, \vec{c_i}))$ dont l'évaluation heuristique totale $\vec{f}(i, \vec{c_i})$ est non dominée au sein de $OPEN$. Une fois le triplet ouvert sélectionné, le chemin partiel associé est étendu en visitant chaque successeur j du sommet i.

Dès qu'un nouveau chemin partiel est identifié en un sommet j, les chemins partiels stockés dans $C_{open}(j)$ et $C_{closed}(j)$ peuvent être élagués, à la suite de quoi la liste globale $OPEN$ est actualisée (les triplets associés aux chemins partiels supprimés sont à leur tour supprimés). On rappelle que cet élagage est possible car le principe d'optimalité est valide.

Au fur et à mesure, des chemins solution de o vers d vont être identifiés ; leurs coûts totaux sont stockés dans la liste $COSTS$. Dès qu'un nouveau chemin intégral p_{od} est identifié, son vecteur coût $\vec{c}(p_{od})$ est utilisé pour *filtrer* les triplets dans $OPEN$ pour lesquels $\vec{c}(p_{od}) \prec \vec{f}$ (filtrage 1). Par ailleurs, avant qu'un nouveau triplet $(i, \vec{c_i}, \vec{f}(i, \vec{c_i}))$ ne soit ajouté à $OPEN$, il doit être vérifié que son estimation heuristique n'est pas dominée par un vecteur coût dans $COSTS$ (filtrages 2 et 3).

La recherche se termine lorsque la liste $OPEN$ est vide. Ainsi, le destin de chaque triplet dans OPEN est soit d'être sélectionné pour être exploré, soit d'être filtré, soit d'être élagué.

Suite à cela, les chemins pareto-optimaux peuvent être reconstitués grâce aux pointeurs vers les nœuds parents étiquetés aux vecteurs coûts des listes C_{closed}. La construction des chemins pareto-optimaux se fait en partant de $C_{closed}(d)$. Dans certains cas, le nombre de chemins solutions peut croître exponentiellement avec le nombre de nœuds du graphe.

6 NAMOA*-T : extension de NAMOA* aux valuations dynamiques

Nous proposons une extension de NAMOA* au cas dynamique ; ce nouvel algorithme, que l'on appellera NAMOA*-T, est détaillé dans la fig. 3. Il s'appuie sur les 3 propriétés suivantes (présentées dans la section 4.1) :

1. l'attente au niveau d'un sommet n'est pas permise (du fait des caractéristiques des moteurs de bateau) ;
2. le graphe est FIFO ;
3. le principe d'optimalité n'est pas conservé.

Un algorithme simpliste consiste à explorer tous les chemins possibles dans le graphe à partir du sommet origine o, en arrêtant l'exploration d'une chemin lorsque sa durée totale dépasse une certaine date $t_{max} \in T$ qui est la plus grande valeur de l'ensemble T (sans cette condition d'arrêt, l'algorithme peut ne pas se terminer du fait des circuits dans le graphe). Tout les chemins explorés peuvent être stockés dans un ensemble duquel sont extraits les chemins pareto-optimaux, solutions du problème. Si l'attente au niveau d'un sommet avait été envisagée dans notre modèle, il aurait été nécessaire d'ajouter des arcs au graphe (d'un sommet vers lui même), et d'imposer une durée maximale de trajet. Cependant, cela risquait d'augmenter fortement le nombre d'itérations de l'algorithme. Par ailleurs, si le graphe n'était pas FIFO, étant donné que nous travaillons sur un graphe avec circuits, il pourrait exister des circuits absorbants, c'est-à-dire des circuits qui feraient descendre l'un des critères que l'on cherche à optimiser. Le graphe étant FIFO, il n'existe pas de circuit absorbant, et de ce fait, l'algorithme simpliste proposé est admissible.

NAMOA*-T a pour objectif de tirer partie de l'efficacité de NAMOA*, celle-ci résidant dans deux types de procédures : la procédure d'élagage et la procédure de filtrage. Concernant la procédure d'élagage, elle se base sur le principe d'optimalité, qui comme cela a été

Données : Un graphe $G = (N, A, T)$, deux nœuds $o, d \in N$, deux fonctions c et h, une date de départ t_0

Résultat : Ensemble des chemins non dominés P^*_{od}

```
/* ----------INITIALISATION---------                    */
```
Création pour tout $i \in N$ des listes $C_{open}(i)$ et $C_{closed}(i)$
Création d'une liste de triplets $OPEN$
Création d'une liste de coûts finaux $COSTS$
$\vec{c_o} = \vec{0}$
$\vec{f}(o, \vec{c_o}) = \vec{c_o} + \vec{h_o}$
$OPEN \leftarrow (o, \vec{c_o}, \vec{f}(o, \vec{c_o}))$
$C_{open}(o) \leftarrow \vec{c_o}$

```
/* -----------ITÉRATIONS-----------                     */
```
tant que $OPEN$ *non vide* **faire**
  Sélection d'un triplet non dominé $(i, \vec{c_i}, \vec{f}(i, \vec{c_i})) \in OPEN$
  Supprimer $(i, \vec{c_i}, \vec{f}(i, \vec{c_i}))$ de $OPEN$
  Déplacer $\vec{c_i}$ de $C_{open}(i)$ vers $C_{closed}(i)$
  si i=d **alors**
    $COSTS \leftarrow \vec{c_i}$
    update($OPEN$) `// Filtrage 1`
  sinon
    pour $j \in$ *successeurs de* i **faire**
      $t' = t_0 + [\vec{c_i}]_1$
      $\vec{c_j} = \vec{c_i} + \vec{c}_{ij}(t')$
      si j *est un nouveau nœud* **alors**
        $\vec{f}(j, \vec{c_j}) = \vec{c_j} + \vec{h_j}$
        si $COSTS \not\prec \vec{f}(j, \vec{c_j})$ `// Filtrage 2` **alors**
          $OPEN \leftarrow (j, \vec{c_j}, \vec{f}(j, \vec{c_j}))$
          $C_{open}(j) \leftarrow \vec{c_j}$
          Étiqueter à $\vec{c_j}$ un pointeur vers i
        fin
      sinon si $\vec{c_j} \in \left[C_{open}(j) \cup C_{closed}(j)\right]$ **alors**
        Étiqueter à $\vec{c_j}$ un pointeur vers i
      sinon
        `// -------------------------`
        `// Absence de phase d'élagage`
        `// -------------------------`
        $\vec{f}(j, \vec{c_j}) = \vec{c_j} + \vec{h_j}$
        si $COSTS \not\prec \vec{f}(j, \vec{c_j})$ `// Filtrage 3` **alors**
          $OPEN \leftarrow (j, \vec{c_j}, \vec{f}(j, \vec{c_j}))$
          $C_{open}(j) \leftarrow \vec{c_j}$
          Étiqueter à $\vec{c_j}$ un pointeur vers i
        fin
      fin
    fin
  fin
fin

```
/* ---CONSTRUCTION DE LA SOLUTION---                    */
```
Construction de P^*_{od} à partir de $C_{closed}(d)$

FIG. 3 – NAMOA*-T

démontré dans la partie 4.1 n'est plus valide. Ainsi, la procédure d'élagage est supprimée dans NAMOA*-T. Quant à la procédure de filtrage, elle est toujours valide, ce qui est démontré par la suite.

Théorème 2. *L'utilisation de la procédure de filtrage ne modifie pas l'admissibilité de NAMOA*-T.*

Preuve 2. *Soit p_{oi} un chemin de o vers i dans G quittant le nœud o à la date t_0, avec $\vec{c_i} = \vec{c}(p_{oi})$. L'estimation heuristique $\vec{h_i}$ associée à ce sommet ne dépend pas du temps. Cette estimation est optimiste ce qui implique (cf. Définition 6) que $\vec{h_i} \prec \overrightarrow{MIN}_i$.*

Le filtrage se fait sur le coût estimé $\vec{f}(i, \vec{c_i}) = \vec{c_i} + \vec{h_i}$. S'il existe un chemin p'_{od} quittant le nœud o à t_0 tel que :

$$\vec{c}(p'_{od}) \prec \vec{f}(i, \vec{c_i}) \tag{4}$$

autrement dit :

$$\vec{c}(p'_{od}) \prec \vec{c_i} + \vec{h_i} \tag{5}$$

alors, par transitivité de la relation de dominance on a :

$$\vec{c}(p'_{od}) \prec \vec{c_i} + \overrightarrow{MIN}_i \tag{6}$$

Cela signifie que les meilleurs chemins de o vers d constitués du chemin partiel p_{oi} sont dominés par une solution existante p'_{od}. De ce fait, le chemin partiel p_{oi} ne peut appartenir à une solution non dominée et peut donc être éliminé sans remettre en cause l'admissibilité de l'algorithme. □

La preuve se rapproche de son homologue dans le contexte des graphes statiques, d'autant plus que dans notre notation la notion de temps est encapsulée dans le vecteur coût.

Finalement, l'algorithme simpliste agrémenté de la procédure de filtrage est admissible. C'est cet algorithme qui est détaillé dans la fig. 3, et qui présente les adaptations suivantes par rapport à NAMOA* :

- les données en entrée sont liées au temps ;
- la procédure d'élagage est supprimée ;
- l'accès au coût d'un arc (i, j) dépend de la date à laquelle le nœud i est quitté, accessible via le premier élément du vecteur de coût $\vec{c_i}$.

Le détail du déroulement de l'algorithme sur un exemple jouet est proposé en annexe du rapport de recherche disponible sur le site du LSIS (*www.lsis.org*). On peut voir que les circuits du graphe sont à l'origine de l'exploration de boucles qui ne sont pas identifiées directement par l'algorithme, mais qui disparaissent naturellement avec la prodcédure de filtrage. Il sera intéressant de réfléchir à des procédures permettant d'éviter ces boucles, afin d'accélérer l'algorithme.

7 Conclusion

Dans cet article, nous avons abordé le problème d'optimisation des routes maritimes en fonction des conditions météorologiques en temps réel. La difficulté de ce problème est liée d'une part à son aspect multiobjectif, et d'autre part, à l'existence de valuations dynamiques dans le graphe dont il faut calculer un plus court chemin. Ce problème recèle ainsi deux difficultés qui rendent la question significativement plus complexe, sachant que les résultats obtenus jusqu'à présent ne semblent pas satisfaisants sur le plan pratique. Pour le traiter, nous avons proposé une extension au cas des graphes à valuations dynamiques de l'algorithme NAMOA* qui traite du calcul de plus court chemin multiobjectif mais dans un cadre seulement statique.

L'implémentation de cet algorithme et son intégration au logiciel de routage développé par Atos est en cours. Les résultats seront comparés à un algorithme déjà implémenté, tel que celui de Hamacher. Il est nécessaire que le temps de calcul de l'algorithme soit raisonnable (de l'ordre de quelques minutes tout au plus). La suppression de la procédure d'élagage diminuant l'efficacité de l'algorithme, il sera peut-être nécessaire de réfléchir à des améliorations permettant d'accélérer l'algorithme afin d'identifier des solutions plus rapidement (par exemple en mettant en place une procédure permettant d'éviter le boucles).

Enfin, il sera pertinent dans un deuxième temps d'étudier les algorithmes fournissant un sous-ensemble des solutions pareto-optimales.

Références

[1] *Review of Maritime Transport.* UNCTAD, 2013.

[2] K.A. Andersen A.J. Skriver. *A label correcting approach for solving bicriterion shortest-path problems,*, 27 :507–524, 2000.

[3] Richard Bellman. On a routing problem. *Quarterly Applied Mathematic*, 16 :87–90, 1958. 4.

[4] I. Chabini. Discrete dynamic shortest path problems in transportation applications,. *Transportation Research Record*, 8 :170–175, 1998.

[5] J.N. Clímaco and E.Q. Martins. On the determination of the nondominated paths in a multiobjective network problem. *Operations Research*, 40 :255–258, 1981.

[6] Kenneth L Cooke and Eric Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3) :493 – 498, 1966.

[7] H. Corley and I. Moon. *Journal of Optimization Theory and Applications*, 46 :79–86, 1985.

[8] S. E. Dreyfus. An appraisal of some shortest path algorithms. 17(3) :395–412, 1969. 4.

[9] J.L. Santos E.Q. Martins. An algorithm for the quickest path problem,. *Operations Research Letters*, 20 :195–198, 1997.

[10] Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. 5.

[11] A.M. Geoffrion. Solving bicriterion mathematical programs. *Operation research*, 15(1) :39–54, 1967.

[12] Tristram Gräbener, Alain Berro, and Yves Duthen. Time dependent multiobjective best path for multimodal urban routing. *Electronic Notes in Discrete Mathematics*, 36 :487 – 494, 2010. 8.

[13] Horst W. Hamacher, Stefan Ruzika, and Stevanus A. Tjandra. Algorithms for time-dependent bicriteria shortest path problems. *Discrete Optimization*, 3(3) :238 – 254, 2006. 9.

[14] P. Hansen. Bicriterion path problems. In Günter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127. Springer Berlin Heidelberg, 1980. 8.

[15] D. Shier J. Brumbaugh-Smith. An empirical investigation of some bicriterion shortest path algorithms,. *European Journal of Operational Research*, 43 :216–224, 1989.

[16] J.M.J. Journée and J.H.C. Meijers. Ship routeing for optimum performance. *Transactions IME*, February 1980. 5.

[17] E. Kanoulas, Y. Du ane T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. *Data Engineering*, 2006.

[18] D.E. Kaufman and R.L. Smith. Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. *Journal of Intelligent Transportation Systems*, 1 :1–11, 1993.

[19] Michael M. Kostreva and Malgorzata M. Wicek. Time dependency in multiple objective dynamic programming. *Journal of Mathematical Analysis and Applications*, 173(1) :289–307, February 1991. 7.

[20] X. Gandibleux M. Ehrgott. *Multiple Criteria Optimization*. State of the Art Annotated Bibliographic Surveys, Boston, 2002.

[21] L. Mandow and J.L. Perez De La Cruz. A new approach to multiobjective a* search. In *Proceedings of the XIX International Joint Conference on Artificial Intelligence*, 2005. 8.

[22] E.Q. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16 :236–245, 1984.

[23] John Mote, Ishwar Murthy, and David L. Olson. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research*, 53(1) :81 – 92, 1991. 8.

[24] Ariel Orda and Raphael Rom. Shortest-path and minimum -delay algorithms in networks with time-dependent edge-length. 37(3) :607–625, February 1990. 4.

[25] Gaël Sauvanet. *Recherche de chemins multiobjectifs pour la conception et la réalisation d'une centrale de mobilité destinée aux cyclistes*. Theses, Université François Rabelais - Tours, April 2011.

[26] Antonio Sedeño-Noda and Andrea Raith. A dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem. *Computers & Operations Research*, 57 :83 – 94, 2015. 8.

[27] L. Lancaster T. Getachew, M. Kostreva. A generalization of dynamic programming for pareto optimization in dynamic networks,. *Operation Research*, 34 :27–47, 2000.

[28] Aphrodite Veneti, Charalampos Konstantopoulos, and Grammati Pantziou. Continuous and discrete time label setting algorithms for the time dependent bi-criteria shortest path problem. *Computing Society Conference*, pages 62–73, 2015. 8.

[29] P. Vincke. Problèmes multicritères. *Cahiers du Centre d'Etudes de Recherche Opérationelle*, (16) :425—439, 1974. 6.

Vers une décomposition dynamique des CSP

Philippe Jégou **Hanan Kanso** **Cyril Terrioux**
Aix-Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296
13397 Marseille Cedex 20 (France)
{philippe.jegou, hanan.kanso, cyril.terrioux}@lsis.org

Résumé

Dans cet article, nous traitons deux questions clés relatives aux méthodes de résolution basées sur la décomposition arborescente de CSP. Tout d'abord, nous proposons un algorithme de calcul de décompositions qui permet de borner la taille des séparateurs, soit un paramètre crucial pour limiter la complexité en espace, et donc la faisabilité pratique de ce type de méthodes. Ensuite, nous montrons comment il est possible de modifier une décomposition dynamiquement pendant une résolution. Cette modification dynamique permet d'offrir plus de liberté aux heuristiques de choix de variables. Cela permet aussi un meilleur usage des informations obtenues pendant la résolution tout en contrôlant la taille de l'espace mémoire requis.

Abstract

In this paper, we address two key aspects of solving methods based on tree-decomposition. First, we propose an algorithm computing decompositions that allows to bound the size of separators, which is a crucial parameter to limit the space complexity, and thus the feasibility of such methods. Moreover, we show how it is possible to dynamically modify the considered decomposition during the search. This dynamic modification can offer more freedom to the variable ordering heuristics. This also allows to better use the information gained during the search while controlling the size of the required memory.

1 Introduction

Les méthodes de résolution basées sur la décomposition arborescente de CSP ont démontré leur intérêt théorique car elle permettent de garantir des bornes de complexité en temps en $O(exp(w))$ aussi bien qu'en espace en $O(exp(s))$ où w et s sont des paramètres induits par des propriétés structurelles des réseaux de contraintes. Ainsi, quand w est borné par une constante, ces méthodes assurent un temps de résolution polynomial. De plus, en pratique, ces approches sont particulièrement justifiées car il existe de nombreux problèmes réels pour lesquels w est relativement petit [6]. Cependant, deux problèmes majeurs se présentent souvent en pratique. Tout d'abord, le contrôle de la valeur de s n'est pas toujours garanti, en particulier pour des algorithmes de calcul de décompositions tels que $Min\text{-}Fill$ [22] qui peut cependant être considéré comme l'état de l'art [7]. Cela rend ce type d'approches parfois complètement inopérant en pratique car sur ce plan, ce paramètre est crucial [1]. De plus, assurer une complexité en temps en $O(exp(w))$ requiert un parcours de l'espace de recherche qui impose de fortes contraintes sur l'ordre d'affectation des variables, ce qui peut conduire à une très forte détérioration de l'efficacité pratique. Pour répondre à la question relative à l'espace mémoire, nous proposons un nouvel algorithme de calcul de décompositions qui est paramétrable. Il prend en entrée un paramètre S et va permettre un calcul de décomposition qui garantie une taille de séparateurs inférieure ou égale à S. Sa complexité en temps est inférieure à celle de $Min\text{-}Fill$ et il offre de plus des performances pratiques qui sont largement meilleures (environ 1000 fois plus rapide sur un large ensemble de benchmarks). La deuxième partie de l'article propose une approche qui sert à modifier dynamiquement une décomposition pendant une recherche, permettant ainsi d'offrir plus de liberté aux heuristiques tout en poursuivant l'exploitation des décompositions. Cette approche est motivée par le fait que pour être efficace en pratique, il est en général nécessaire de prendre en compte le contexte courant d'une recherche ainsi que les connaissances acquises progressivement pendant la résolution. Ce type d'approche existe au niveau des solveurs CSP qui utilisent des heuristiques basées sur des pondérations attachées aux contraintes comme par exemple pour *dom/wdeg* [4]. Elle est présente également au niveau des solveurs CDCL pour SAT [19] par le biais notamment des techniques d'apprentissage de clauses et de redémarrage.

Dans le cas des méthodes opérant par décomposition, la difficulté fondamentale porte sur l'ordre d'affectation des variables qui est imposé par la décomposition. Pour contourner cette difficulté, nous proposons d'adapter dynamiquement la décomposition pendant la résolution. La première idée concernant une telle approche a été introduite dans [11] mais principalement sur un plan théorique. Aussi, nous montrons comment une telle approche est finalement réalisable. Par ailleurs, nous étendons cette approche en intégrant les techniques de redémarrage comme proposé par [13]. De plus, nous décrivons comment modifier dynamiquement une décomposition, en prenant en compte les connaissances acquises pendant la résolution tout en proposant de conserver une borne sur la taille des séparateurs pendant la recherche.

La section 2 rappelle des notions sur les méthodes de résolution basées sur les décompositions arborescentes tandis que la section 3 présente un algorithme de calcul de décompositions arborescentes qui prend en compte la taille des séparateurs. La section 4 introduit une extension de l'algorithme BTD [12] visant à adapter la décomposition pendant la résolution. Enfin, avant de conclure, la section 5 présente des expérimentations pour évaluer la pertinence de cette approche.

2 Préliminaires

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, 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*. L'*arité* d'une contrainte c_i est $|S(c_i)|$. Si l'arité de chaque contrainte est égale à 2, on parle alors de CSP *binaire*. La structure d'une instance CSP est donnée par un hypergraphe (un graphe pour les CSP binaires), appelé *(hyper)graphe* de contraintes, dont les sommets correspondent aux variables et les arêtes aux portées des contraintes. Pour simplifier les notations, nous notons l'hypergraphe $(X, \{S(c_1), \ldots S(c_e)\})$ par (X, C). Une affectation d'un sous-ensemble de X est dite *cohérente* si toutes les contraintes portant sur ce sous-ensemble sont satisfaites. Le problème du test d'existence d'une *solution* (i.e. une affectation cohérente de X) est bien connu pour être NP-complet. Aussi, de nombreux travaux ont été réalisés pour améliorer la résolution pratique comme ceux portant sur les heuristiques de choix de variables, l'apprentissage de contraintes, le backtracking non-chronologique, ou encore les techniques de filtrage. Néanmoins, la complexité de la résolution demeure exponentielle, au moins en $O(n.d^n)$ où d est la taille maximum des domaines. Pour contourner cette « intraitabilité » théorique, d'autres approches ont été proposées. Certaines d'elles s'appuient sur la notion de classes polynomiales structurelles [8] basées sur la notion de *décomposition arborescente de graphes* [21].

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, F est un ensemble d'arêtes T) 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 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 un chemin de i à 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.*

Cette notion est seulement définie pour des graphes mais elle peut être considérée pour des hypergraphes en exploitant leur *2-section*[1]. L'avantage des méthodes de résolution exploitant les décompositions arborescentes est d'abord lié à leur complexité théorique en temps d^{w+1} [7] alors que la complexité en espace est en d^s où s est la taille maximum des intersections (appelés *séparateurs* dans la suite) entre clusters. Ainsi, ces méthodes peuvent être efficaces sur des instances de grande taille et de petite tree-width comme c'est le cas, par exemple, pour les problèmes d'allocation de fréquences radio [5]. Ces méthodes procèdent en deux étapes : (1) calcul d'une décomposition arborescente, et (2) résolution d'une instance exploitant la décomposition. Puisque le calcul de décompositions optimales (i.e. de largeur w) est NP-difficile [2], en pratique, la première étape calcule en général des décompositions arborescentes de largeur $w^+ \geq w$, à savoir une approximation de la tree-width. Dans ce contexte, la méthode $Min\text{-}Fill$ [22] apparaît comme le meilleur compromis entre le temps de calcul ($O(n^3)$) et la qualité des décompositions obtenues. Elle constitue la référence dans l'état de l'art pour de tels algorithmes [7], même si, pour des graphes de quelques dizaines de milliers de sommets, elle est en général inutilisable en pratique.

Cependant, les décompositions calculées ne sont pas nécessairement les plus adaptées du point de vue de la résolution [10, 14]. En premier lieu, $Min\text{-}Fill$ ne prend pas en compte explicitement les propriétés structurelles des graphes considérés, ce qui peut rendre la résolution inefficace. Par exemple, les décompositions obtenues peuvent contenir des clusters non connexes ce qui peut conduire à une forte dégradation de l'efficacité

1. La 2-section d'un hypergraphe (X, C) est le graphe (X, C') où $C' = \{\{x, y\} | \exists c \in C, \{x, y\} \subseteq c\}$ [3].

de la résolution [14]. De plus, *Min-Fill* peut générer des décompositions telles que s est souvent proche de w^+. En effet, afin de minimiser la largeur, *Min-Fill* produit des clusters avec peu de sommets *propres* (i.e. des sommets appartenant à un cluster mais pas à son cluster parent dans la décomposition), voire même, un seul sommet propre. Cela peut mener à un coût qui s'avère prohibitif en termes d'espace mémoire. Ainsi, la minimisation de s est cruciale pour être efficace en pratique [10]. Par ailleurs, pour garantir la complexité en temps d^{w^+}, les méthodes structurelles efficaces comme BTD [12] utilisent un ordre d'affectation des variables qui est partiellement déterminé par la décomposition considérée. Quand *Min-Fill* est utilisé, cette liberté est même encore plus restreinte du fait du nombre limité de sommets propres dans les clusters. Or, il est bien connu que pour avoir une résolution efficace, il est souhaitable d'avoir une liberté maximale pour le choix des prochaines variables à affecter.

Pour contourner ces difficultés, plusieurs approches sont possibles. La première consiste à s'appuyer sur une décomposition disposant de petits séparateurs, tout en ayant de plus grands clusters. Cela permet de relâcher les contraintes portant sur l'ordre d'affectation [10]. Une autre possibilité porte sur l'exploitation des redémarrages comme proposé dans [13]. Cette approche fonctionne en redémarrant la recherche à partir d'une nouvelle première variable qui ne figure pas nécessairement dans le précédent cluster racine. Ceci conduit, tout en conservant la même décomposition (excepté en termes de racine), à donner plus de liberté à l'ordre. La pertinence d'une telle démarche a été démontrée expérimentalement dans [13]. Une autre possibilité consiste à modifier dynamiquement la décomposition pendant la résolution, tout en maintenant des garanties au niveau de la complexité temporelle. Cette approche a été introduite dans [11], mais principalement sur un plan théorique. Elle propose d'étendre la taille d'un cluster en le fusionnant avec un ou plusieurs clusters voisins. Toutefois, sa pertinence n'a jamais véritablement été démontrée. De plus, telle qu'elle a été introduite dans [11], elle n'est guidée que par des critères d'ordre structurel, sans prendre en compte explicitement l'état de la recherche, et surtout, les connaissances déjà acquises pendant la résolution. Notons que l'exploitation de la structure des instances de manière dynamique a déjà été proposée pour SAT dans [17]. Cela étant, dans ces travaux, aucune garantie ne pouvait être donnée au sujet des bornes de complexité, contrairement à ce que nous proposons ici.

Pour proposer des voies alternatives à ces précédents travaux, nous introduisons dans la suite d'abord un algorithme qui calcule des décompositions dont la taille des séparateurs est bornée. Ensuite, nous proposons un nouvel algorithme de résolution basé sur BTD et qui permet d'adapter dynamiquement les décompositions pendant la recherche en exploitant des informations obtenues durant la résolution.

3 Décomposer en bornant les séparateurs

Dans [9], nous avons proposé un cadre général pour calculer des décompositions arborescentes qui ne repose pas sur la notion de triangulation. Il permet d'implémenter différentes méthodes de décomposition selon les caractéristiques souhaitées pour la décomposition. Par exemple, l'algorithme *BC-TD* [14], qui calcule des décompositions dont tous les clusters sont connexes, est une variante possible de ce cadre. Avec la définition de ce cadre, nos objectifs sont multiples. D'abord, pour pouvoir traiter dynamiquement des décompositions, il faut disposer d'algorithmes efficaces, tant en termes de complexité théorique, que d'efficacité pratique. Ensuite, la complexité de ces algorithmes ne doit pas dépasser $O(n(n+e))$ afin d'être meilleure que celle de *Min-Fill*, ce qui peut être accompli en économisant une étape des plus coûteuses, à savoir la triangulation. Au-delà, il faut veiller à limiter la taille maximale des clusters, ce qui reste et demeure un paramètre important, tout en maîtrisant la taille des séparateurs.

Dans cette partie, nous rappelons le cadre *H-TD-WT* (pour *Heuristic Tree-Decomposition Without Triangulation*) qui calcule une décomposition arborescente du graphe $G = (X, C)$ sans triangulation en temps polynomial (à savoir en $O(n(n + e))$). Comme pour *Min-Fill*, aucune garantie n'est offerte quant à l'optimalité de la largeur obtenue. Tel que présenté dans [9], ce cadre permet cependant différentes paramétrisations en prenant en compte plusieurs critères dont certains sont relatifs à w^+ et/ou s. Nous nous intéressons ici à maîtriser la taille des séparateurs.

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 $X_1, X_2, \ldots X_k$ les composantes connexes du sous-graphe $G[X \backslash E_0]$ induit par la suppression dans G des sommets de E_0 [2]. 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). Dans l'exemple de la figure 1, pour $X_i = X_1$, on a $V_i = V_1 = \{x, y, z\}$.

On peut constater que V_i est 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)

2. Le sous-graphe $G[Y]$ de $G = (X, C)$ induit par $Y \subseteq X$ est le graphe (Y, C_Y) où $C_Y = \{\{x, y\} \in C | x, y \in Y\}$.

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 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$

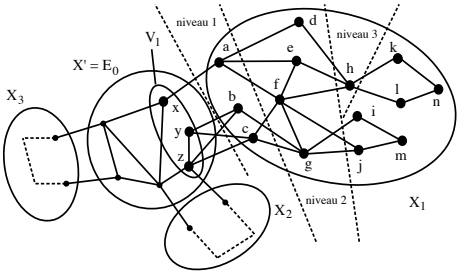


FIGURE 1 – Graphe illustrant H_5.

peut être paramétrée. 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.

Dans [9], ce schéma était décliné selon quatre heuristiques. La première (notée H_1) vise à minimiser la taille des clusters alors que la seconde (H_2) garantit que tous les clusters sont connexes (voir [14]). La troisième (H_3) tente d'identifier 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. La quatrième (H_4) tend à limiter la taille des séparateurs de la décomposition. Pour cela, elle considère un paramètre S qui représente la taille maximale autorisée pour un séparateur. Elle ajoute de nouveaux sommets au prochain cluster E_i de la même manière que H_3. Toutefois, la recherche est stoppée à un niveau $l = L$ dès que les toutes les composantes connexes de $G[X_i \backslash E_{i_L}]$ induites sont déconnectées du reste du graphe par des séparateurs de taille au plus S.

Nous introduisons ici une nouvelle heuristique, naturellement notée H_5. Elle vise à raffiner l'heuristique H_4 en détectant plus de séparateurs de taille au plus S. Là où H_4 doit atteindre un niveau de la recherche en largeur à partir duquel tous les séparateurs sont de taille au plus S, H_5 va être plus opportuniste. Si à un niveau donné, une nouvelle composante connexe apparaît avec un séparateur de taille au plus S, ce séparateur va être pris en compte. Sa composante connexe sera rajoutée à la file d'attente, et la recherche va se poursuivre sur le reste du sous-graphe en cours d'exploration, exceptée bien sûr, la partie associée à cette nouvelle composante connexe.

Nous illustrons le fonctionnement de H_5 sur l'exemple de la figure 1. Nous évoquons en premier le calcul de E_1, le second cluster (après E_0) lors du premier passage dans la boucle et nous fixons la valeur du paramètre S à 2. On considère donc l'ensemble $V_1 = \{x, y, z\}$. Les sommets a, b et c constituent le premier niveau. Aucun sous-ensemble de $\{a, b, c\}$ de taille 2 ou moins n'induit de séparateur. Le niveau suivant, constitué des sommets d, e, f et g, est donc visité. À ce niveau, on dispose de 2 séparateurs minimaux, d'une part $\{d, e, f\}$ et d'autre part $\{f, g\}$. Avec l'heuristique H_4, la recherche en largeur se poursuivrait. Avec H_5, elle va certes se poursuivre, mais l'existence du séparateur $\{f, g\}$ est exploitée. Cela va permettre, non pas d'identifier un nouveau cluster, mais de circonscrire la poursuite de la recherche en largeur à un sous-graphe duquel auront été enlevés les sommets i, j et m car il ne sont plus connectés au reste des autres sommets. Ainsi, l'ensemble $\{i, j, m\}$ va être inséré dans la fille d'attente F. Comme ceci ne figure pas explicitement dans le schéma de l'algorithme, il faut considérer que cet ajout fait partie intégrante du traitement réalisé à la ligne 8. La recherche en profondeur se poursuit donc mais sur la partie de X_1 qui n'a pas encore été parcourue, et dont $\{i, j, m\}$ a été supprimé, soit $\{h, k, l, n\}$. On constate alors que le niveau suivant, uniquement constitué du sommet h constitue un séparateur de taille 1, donc inférieur à la borne fixée $S = 2$. Le parcours en largeur est stoppé, et le nouveau cluster est maintenant constitué : $E_1 = \{x, y, z, a, b, c, d, e, f, g, h\}$ et $X_1'' = \{a, b, c, d, e, f, g, h\}$. On obtient une composante connexe $X_{1_1} = \{k, l, n\}$ qui est ajoutée à F.

En fait, si on revient au schéma général de l'algorithme *H-TD-WT*, la version associée à l'heuristique H_5 respecte bien les conditions imposées par l'approche, à savoir que l'on construit bien un sous-ensemble $X_i'' \subseteq X_i$ tel qu'il existe au moins un sommet $v \in V_i$ tel que $N(v, X_i) \subseteq X_i''$. Cette observation, conjuguée à la preuve figurant dans [9], permet de s'assurer de la validité de H_5 :

Théorème 1 *H_5 calcule les clusters d'une décomposition arborescente.*

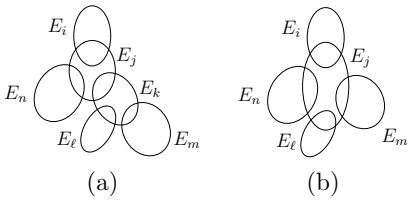


FIGURE 2 – Ensemble de clusters d'une décomposition avant fusion de E_k dans E_j (a) et après (b).

Il en est de même que pour l'analyse de sa complexité.

Théorème 2 *La complexité en temps de l'algorithme H_5-TD-WT est $O(n(n+e))$.*

4 Décomposition dynamique

La thèse que nous défendons ici est que la dynamicité de la décomposition, i.e. sa modification pendant la résolution, permet d'adapter la décomposition à la nature de l'instance à résoudre. Pour modifier la décomposition, on s'appuie sur des informations recueillies pendant la résolution, notamment celles liées à la sémantique du problème. Cette démarche s'inscrit alors dans la lignée des méthodes dites *adaptatives*. Ces méthodes font des choix non seulement par rapport à l'état courant de la résolution, mais aussi par rapport aux états précédents. Ce faisant, elles ont démontré leur intérêt pratique (comme dans [4, 18, 20]) par rapport aux méthodes classiques. Par exemple, en utilisant des stratégies de choix de variables orientées vers les conflits, on peut identifier les variables les plus problématiques pendant la recherche en enregistrant en permanence des informations sur les conflits rencontrés. Ainsi, cela permet de considérer ces variables plus tôt dans le reste de l'arbre de recherche et, par là même, de résoudre l'instance plus efficacement. Si l'ordre sur les variables est totalement libre, la variable suivante serait donc la variable jugée comme la plus influente pour la suite de la résolution. Par contre, pour des méthodes exploitant une décomposition arborescente comme BTD, l'ordre sur les variables étant partiellement induit par la décomposition, la liberté de l'heuristique de choix de variables s'en trouve d'autant restreinte. Pour y remédier, nous proposons d'adapter la décomposition durant la résolution en procédant à une fusion dynamique de certains clusters.

L'algorithme BTD-MAC+RST+Fusion (voir l'algorithme 3) représente une adaptation de l'algorithme BTD-MAC+RST [13] afin de prendre en compte la fusion dynamique. Pour ces deux algorithmes, l'emploi d'une décomposition arborescente avec un certain cluster racine E_r induit un ordre partiel sur les variables. Si E_j est le cluster courant, le choix de variables se limite alors soit à choisir parmi les variables non encore instanciées du cluster E_j, soit à choisir un prochain cluster parmi les fils de E_j une fois le cluster E_j entièrement instancié. Les deux algorithmes débutent la résolution avec une décomposition calculée en amont de celle-ci. La différence principale entre eux réside dans le fait que BTD-MAC+RST n'utilise que cette décomposition initiale durant toute la résolution (au sens de l'ensemble des clusters qui la définissent puisque la racine peut changer), tandis que BTD-MAC+RST+Fusion va la faire évoluer dynamiquement. Ainsi, l'ordre partiel imposé par la décomposition change pendant la résolution. L'opération per-

Algorithme 2 : BTD-MAC+Fusion

```
   Entrées-Sorties : P = (X, D, C) : CSP
   Entrées : Σ : suite de décisions ; E_i : cluster ; V_{E_i} : ensemble de
             variables
   Entrées-Sorties : G : ensemble de goods ; N : ensemble de nogoods
   Sorties : vrai si une solution au sous-problème de P enraciné en E_i et
             induit par Σ a été trouvée, faux s'il est prouvé qu'il n'en
             possède pas, inconnu sinon
1  si V_{E_i} = ∅ alors
2      résultat ← vrai
3      S ← Fils(E_i) /* Fils(E_i) : ensemble des clusters fils de E_i */
4      tant que résultat ∉ {faux, inconnu} et S ≠ ∅ faire
5          Choisir un cluster E_j ∈ S
6          S ← S\{E_j}
7          si Pos(Σ)[E_i ∩ E_j] est un nogood dans N alors
8              résultat ← faux
9          sinon
10             si Pos(Σ)[E_i ∩ E_j] n'est pas un good de E_i par rapport à E_j
               dans G alors
11                 résultat ← BTD-MAC+Fusion(P,Σ,E_j,
                   E_j\(E_i ∩ E_j),G,N)
12                 si résultat= vrai alors
13                     Enregistrer Pos(Σ)[E_i ∩ E_j] comme good de
                       E_i par rapport à E_j dans G
14                 sinon
15                     si résultat= faux alors
16                         Enregistrer Pos(Σ)[E_i ∩ E_j] comme
                           nogood de E_i par rapport à E_j dans N
17                     sinon
18                         si fusion alors Fusionner E_j avec un de
                           ses fils
19                         si non redémarrage alors
20                             S ← S ∪ {E_j}
21                             résultat ← vrai
22     retourner résultat
23 sinon
24     Choisir une variable x ∈ V_{E_i}
25     Choisir une valeur v ∈ d_x
26     d_x ← d_x\{v}
27     si AC (P,Σ ∪ ⟨x = v⟩) alors
28         résultat ← BTD-MAC+Fusion(P,Σ∪ ⟨x = v⟩,E_i,
           V_{E_i}\{x},G,N)
29     sinon résultat ← faux
30     si résultat = faux alors
31         si redémarrage ou fusion alors
32             Enregistrer nld-nogoods par rapport à la suite de
               décisions Σ[E_i]
33             retourner inconnu
34         sinon
35             si AC (P,Σ ∪ ⟨x ≠ v⟩) alors
36                 retourner BTD-MAC+Fusion(P,Σ ∪ ⟨x ≠ v⟩,E_i,
                   V_{E_i},G,N)
37             sinon retourner faux
38 sinon retourner résultat
```

Algorithme 3 : BTD-MAC+RST+Fusion

```
   Entrées : P = (X, D, C) : CSP
   Sorties : vrai si P possède une solution, faux sinon
1  G ← ∅ ; N ← ∅
2  répéter
3      Choisir un cluster racine E_r
4      résultat ← BTD-MAC+Fusion (P,∅,E_r,E_r,G,N)
5  jusqu'à result ≠ unknown
6  retourner résultat
```

mettant de changer la décomposition dans ce contexte est la *fusion*. La fusion consiste à mettre en commun les variables de deux clusters voisins pour former ainsi un seul cluster. La figure 2 montre la fusion des clusters E_j et E_k. À noter que, les fils du cluster fusionné deviennent les fils du cluster résultant de la fusion. Par exemple, dans la figure 2, E_l et E_m, les fils de E_k, deviennent les fils de E_j. Soit D la décomposition initiale et D' la décomposition obtenue après la fusion. Tout ordre sur les variables permis par D reste permis par D'. Cependant, en exploitant D' on peut obtenir plus d'ordres possibles qu'en exploitant D. On en déduit que la fusion préserve les ordres de choix de variables initialement permis tout en offrant plus de liberté. Le choix de fusionner ou non des clusters est conditionné par des informations apprises durant la résolution. Aussi, le comportement de BTD-MAC+RST+Fusion se situe entre celui de BTD-MAC+RST avec une liberté partielle pour l'ordre sur les variables (si aucune fusion n'est effectuée) à celui de MAC [23] avec une liberté totale (si, après une série de fusions, la décomposition ne contient plus qu'un seul cluster). L'intérêt de ce nouvel algorithme est donc de pouvoir trouver dynamiquement le bon compromis à partir d'informations recueillies durant la résolution.

L'algorithme BTD-MAC+RST+Fusion exploite l'algorithme BTD-MAC+Fusion (voir l'algorithme 2). Ce dernier ne se différencie de BTD-MAC+NG [13] que par les lignes 17 à 21. Initialement, la suite de décisions Σ ainsi que les ensembles de goods G et de nogoods N sont vides. BTD-MAC+RST+Fusion (à l'instar de BTD-MAC+RST) commence la résolution en assignant les variables du cluster racine E_r avant de passer à un cluster fils. En exploitant le nouveau cluster E_i, seules les variables non assignées du cluster E_i seront instanciées. En d'autres termes, seules les variables de E_i n'appartenant pas à $E_i \cap E_{p(i)}$ (avec $E_{p(i)}$ le cluster père de E_i) sont instanciées. Pour résoudre chaque cluster les deux algorithmes s'appuient sur MAC (lignes 24-29 et 35-37). Durant la résolution MAC peut prendre deux types de décisions : des décisions positives $x_i = v_i$ qui assignent la valeur v_i à la variable x_i et des décisions négatives $x_i \neq v_i$ qui assurent que v_i ne peut être assignée à x_i. Supposons que $\Sigma =< \delta_1, ..., \delta_i >$ est la suite de décisions courante où chaque δ_j peut être une décision soit positive, soit négative. Une nouvelle décision positive $x_{i+1} = v_{i+1}$ est prise et un filtrage par cohérence d'arc (AC) est accompli (ligne 27). Si aucune incohérence n'est détectée, la recherche continue normalement (ligne 28). Sinon la valeur v_{i+1} est supprimée de $d_{x_{i+1}}$ et un nouveau filtrage par AC est effectué (ligne 35). Si une incohérence est détectée, on procède à un retour-arrière et la dernière décision positive $x_\ell = v_\ell$ est changée en $x_\ell \neq v_\ell$. Lorsque le cluster E_i est choisi comme cluster suivant, on sait que la prochaine décision positive implique une variable de E_i. Étant donné que le séparateur $E_i \cap E_{p(i)}$ est instancié, le filtrage par AC impacte uniquement les variables de $Desc(E_i)$ (avec $Desc(E_i)$ l'ensemble de variables appartenant à l'union des descendants E_k de E_i). Une fois le cluster E_i complètement instancié de façon cohérente (ligne 1), chaque sous-problème enraciné en un cluster E_j, fils de E_i, sera résolu (ligne 11). Plus précisément, pour un cluster fils E_j et une suite de décision Σ, on résout le problème enraciné en E_j et induit par $Pos(\Sigma)[E_i \cap E_j]$ (avec $Pos(\Sigma)[E_i \cap E_j]$ l'ensemble des décisions positives impliquant les variables de $E_i \cap E_j$ dans Σ). Si on trouve une extension cohérente de Σ sur $Desc(E_j)$, $Pos(\Sigma)[E_i \cap E_j]$ est enregistré comme *good structurel* (ligne 12-13). Si, au contraire, la résolution montre qu'il n'existe aucune extension cohérente de Σ sur $Desc(E_j)$, $Pos(\Sigma)[E_i \cap E_j]$ est enregistré comme *nogood structurel* (lignes 15-16). Ces (no)good structurels sont utilisés ultérieurement dans la recherche pour éviter certaines redondances (lignes 7-8). Si un redémarrage est déclenché (ligne 31), la résolution est interrompue. La gestion des redémarrages est faite comme dans [13] et s'accompagne de l'enregistrement de nld-nogoods réduits [16] qui permettent de ne pas réexplorer des parties de l'espace de recherche déjà visitées. L'intérêt du redémarrage réside dans l'exploitation des connaissances acquises auparavant via les (no)good structurels et les nld-nogoods réduits [16]. Le déclenchement d'un redémarrage peut être conditionné par des paramètres globaux (portant sur l'ensemble du problème) ou locaux (relatifs au cluster courant) ou aussi une combinaison des deux. Les lignes 17-21 concernent uniquement la fusion dynamique donc BTD-MAC+Fusion. La fusion dynamique vise, comme expliqué ci-dessus, à donner plus de liberté à l'heuristique de choix de variables, et en même temps, à limiter l'impact des défauts potentiels mis en avant dans la section 2. Le choix de fusionner ou non des clusters va se faire sur la base d'un critère s'appuyant sur l'état courant du problème, mais aussi sur tout ou partie des états précédemment rencontrés durant la résolution. Si aucune fusion n'est requise ($fusion$ renvoie $faux$), la résolution se poursuit normalement. Au contraire, si ce critère considère que fusionner le cluster courant avec un de ses fils permettrait de rendre la suite de la résolution plus efficace (par exemple en permettant d'instancier plus tôt certaines variables jouant un rôle clé), $fusion$ renvoie $vrai$ et BTD-MAC+Fusion va modifier la décomposition courante en fusionnant ces deux clusters (ligne 18). Pour cela, on commence par désaffecter les variables instanciées du cluster courant et par enregistrer des nld-nogoods réduits (ligne 32) comme le ferait un redémarrage classique. Une fois revenu dans le cluster parent, on procède

à la fusion. À ce stade (ligne 19), soit on continue à revenir en arrière si *redémarrage* est vrai, soit la recherche se poursuit avec l'exploration d'un fils du cluster parent. Notons que désaffecter les variables du cluster courant avant de le fusionner avec un de ses fils n'est pas une nécessité. Toutefois, ce choix devrait permettre de pouvoir exploiter au plus tôt les variables nouvellement ajoutées dans ce cluster.

L'algorithme BTD-MAC-Fusion est paramétrable notamment par l'heuristique de fusion (nous en proposons une dans la partie expérimentale). Un bon choix pour cette heuristique peut améliorer considérablement la résolution en la rendant plus efficace.

Fusionner deux clusters ne remettant pas en cause la validité des (no)goods structurels et des nld-nogoods, nous pouvons prouver la validité de l'algorithme avant de donner ses complexités en temps et en espace.

Proposition 1 *Soit (E', T') la décomposition arborescente d'un graphe G obtenue à partir de la décomposition (E, T) de G en fusionnant le cluster E_y dans le cluster E_x (avec E_y fils de E_x dans (E, T)). Les (no)good structurels de E_i par rapport à E_j (avec $E_j \neq E_y$) et les nld-nogoods réduits enregistrés vis-à-vis de (E, T) restent valides vis-à-vis de (E', T').*

Théorème 3 *BTD-MAC+RST+Fusion est correct, complet et termine.*

Théorème 4 *BTD-MAC+RST+Fusion a une complexité en temps en $O(R.((n.s^2.e.\log(d) + w'^+.N).d^{w'^+ +2} + n.(w'^+)^2.d))$ et une complexité en espace en $O(n.s.d^s + w'^+.(d+N))$ avec w'^+ la largeur de la décomposition arborescente finale, s la taille de la plus grande intersection $E_i \cap E_j$ de la décomposition initiale, R le nombre de redémarrages et N le nombre de nld-nogoods réduits mémorisés.*

Notons qu'il est possible de limiter l'augmentation de la largeur de la décomposition finale par rapport à la décomposition initiale en utilisant une heuristique de fusion convenable. Dans un tel cas de figure, BTD-MAC+RST+Fusion est proche de l'algorithme BDH [11]. Outre l'absence de redémarrages dans BDH, en pratique, il s'en distingue par des fusions déclenchées dynamiquement sur la base d'informations relatives à la résolution en cours alors que BDH se contente de fusionner des clusters en ne faisant pas croître la taille des clusters au-delà d'une limite fixée au préalable.

5 Évaluation expérimentale

Dans cette section, nous évaluons l'intérêt pratique de l'heuristique H_5 et des décompositions dynamiques.

5.1 Protocole expérimental

En ce qui concerne les décompositions, nous avons retenu *Min-Fill* (en tant qu'heuristique de référence dans la littérature et pour sa bonne approximation de la largeur arborescente), H_2 (pour sa garantie de connexité des clusters), H_3 (dont les clusters ont plusieurs fils) et H_5 (pour sa maîtrise de la taille des séparateurs). Pour H_2, les clusters sont construits en choisissant les sommets de la composante connexe concernée par ordre de degré décroissant jusqu'à ce que le cluster devienne connexe (ce qui correspond à l'heuristique $NV2$ de [14]). Pour H_5, la décomposition est utilisée avec différentes valeurs de S.

La décomposition dynamique repose sur une heuristique de fusion. Cette dernière s'appuie sur l'heuristique de choix de variable pour évaluer la nécessité de fusionner. Plus précisément, étant donné un cluster courant E_j, à chaque fois qu'on choisit la variable suivante dans E_j, on va regarder si cette variable aurait été choisie si l'heuristique de choix de variable avait eu la possibilité de sélectionner une variable parmi toutes les variables non instanciées de $E_j \cup (\bigcup_{E_k \in Fils(E_j)} E_k)$. À chaque fois qu'une variable d'un fils E_k de E_j est préférée à une variable de E_j, un compteur propre à E_k est incrémenté. Lorsque le compteur associé à un fils E_k atteint une certaine limite L (à savoir 100 dans nos expérimentations), on fusionne E_k avec E_j.

Au niveau de la résolution, nous considérons BTD-MAC+RST+Fusion et BTD-MAC+Fusion (c.-à-d. BTD-MAC+RST+Fusion sans redémarrage) pour les méthodes exploitant des décompositions dynamiques, BTD-MAC et BTD-MAC+RST en tant que références pour les méthodes exploitant une décomposition statique, et MAC+RST comme méthode énumérative classique de référence. Nous choisissons comme cluster racine le cluster ayant le plus grand rapport nombre de contraintes sur la taille du cluster moins un. La cohérence d'arc est appliquée en pré-traitement via $AC3^{rm}$ et durant la résolution via $AC8^{rm}$ [15]. Toutes les méthodes de résolution utilisent l'heuristique dom/wdeg [4] pour choisir la prochaine variable à instancier.

Tous les algorithmes sont implémentés en C++ au sein de notre propre bibliothèque. 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 sélectionné 1 859 les instances CSP de la compétition CSP 2008 [3]. Pour établir cette sélection, nous avons exclu les instances ayant des décompositions triviales (par exemple les instances ayant un graphe complet) ainsi que les instances ayant des contraintes globales (car non prises en compte dans notre bibliothèque). Les instances retenues représentent la majo-

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

Algorithme	*Min-Fill*		H_2		H_3		H_5	
	#rés.	temps	#rés.	temps	#rés.	temps	#rés.	temps
BTD-MAC	1 344	43 272	1 405	31 429	1 466	31 469	1 469	33 564
BTD-MAC+RST	1 495	43 557	1 518	35 042	1 529	30 187	1 543	33 049
BTD-MAC+Fusion	1 481	42 505	1 518	37 440	1 523	35 101	1 534	34 048
BTD-MAC+RST+Fusion	1 544	41 622	1 547	32 547	1 554	33 736	1 567	34 432

TABLE 1 – Nombre d'instances résolues et temps d'exécution en s pour BTD-MAC(+RST) et BTD-MAC(+RST)+Fusion selon la méthode de décomposition utilisée.

Algorithme	*Min-Fill*	H_2	H_3	H_5
BTD-MAC	34 669	18 018	18 951	18 243
BTD-MAC+RST	24 026	17 233	17 758	16 288
BTD-MAC+Fusion	25 238	17 575	18 753	17 837
BTD-MAC+RST+Fusion	23 832	16 803	17 602	15 718

TABLE 2 – Temps d'exécution en s pour BTD-MAC(+RST) et BTD-MAC(+RST)+Fusion selon la méthode de décomposition utilisée pour les 1 234 instances résolues par tous les algorithmes.

rité des familles d'instances. Pour chaque instance, le temps d'exécution (incluant le calcul de la décomposition) est limité à 15 minutes.

5.2 H_5 comparée aux autres décompositions

La table 1 fournit le nombre d'instances résolues et le temps d'exécution cumulé pour chaque algorithme et pour chaque méthode de décomposition. D'abord, nous comparons les méthodes de décomposition du point de vue de l'efficacité de la résolution par BTD-MAC. Concernant le nombre d'instances résolues, BTD-MAC avec H_5 (pour $S = 50$) résout le plus grand nombre d'instances (à savoir 1 469) tandis qu'avec *Min-Fill*, il en résout le moins (à savoir 1 344). Clairement, les méthodes de décomposition visant à minimiser la largeur ne conduisent pas aux résolutions les plus efficaces. D'autres paramètres ont plus d'impacts comme la connexité des clusters (cf. H_2), le nombre de fils des clusters (cf. H_3) et la maîtrise de la taille des séparateurs (cf. H_5). Notons que ces résultats sont cohérents avec ceux de [14, 9]. Au-delà, pour H_5, l'efficacité de la résolution dépend bien sûr de la valeur choisie pour S. Par exemple, BTD-MAC résout plus d'instances pour $S = 15$ (à savoir 1 514). Par contre, le choix de $S = 50$ est plus intéressant quand on exploite des décompositions dynamiques.

Au niveau du temps d'exécution, pour une comparaison plus équitable, nous ne considérons, dans la table 2, que les 1 234 instances résolues par tous les algorithmes. BTD-MAC obtient les meilleurs résultats avec H_5 (et H_2) et les pires avec *Min-Fill*. Nous devons souligner que le calcul des décompositions avec H_i ($i = 2, 3, 5$) est nettement plus rapide qu'avec *Min-Fill*. Par exemple, H_5 (pour $S = 50$) requiert seulement 7 s pour calculer les décompositions des 1,234 instances contre 7,582 s pour *Min-Fill*.

Notons enfin que les observations faites ici pour BTD-MAC restent valides quelle que soit la variante de BTD considérée comme le montrent les tables 1-4.

5.3 Décompositions dynamiques contre statiques

D'abord, si nous comparons BTD-MAC à BTD-MAC+Fusion (resp. BTD-MAC+RST à BTD-MAC+RST+Fusion) dans les tables 1-2, nous pouvons voir que, quelle que soit la méthode de décomposition utilisée, les méthodes exploitant des décompositions dynamiques résolvent plus d'instances que leur pendant exploitant une décomposition statique pour des temps de résolution similaires ou meilleurs. Cela montre bien l'intérêt de fusionner dynamiquement des clusters durant la résolution.

Par ailleurs, le concept de fusion a déjà été exploité de manière statique (par exemple dans [10]) une fois la décomposition initiale calculée via n'importe quelle méthode (comme *Min-Fill*, H_2, H_3 ou H_5). Les tables 3-4 fournissent les résultats correspondants pour BTD-MAC(+RST). Ici, nous limitons la taille des séparateurs en fusionnant avec son père tout cluster dont la taille du séparateur avec son père dépasse une certaine valeur (à savoir 15 dans les tables 3-4). Nous pouvons constater que, pour le nombre d'instances résolues, BTD-MAC+Fusion est significativement meilleur que BTD-MAC tandis que BTD-MAC+RST+Fusion est comparable ou légèrement meilleur que BTD-MAC+RST. À nouveau, l'exploitation des décompositions dynamiques permet d'obtenir parmi les meilleurs résultats. Au-delà, via la fusion dynamique, nous adaptons la décomposition en fonction des connaissances sémantiques apprises durant la résolution alors que la fusion statique ne repose que sur des critères structurels et nécessite de choisir une limite pour la taille des séparateurs, ce qui peut constituer une tâche difficile.

Enfin, nous pouvons remarquer que BTD-MAC+RST et BTD-MAC+Fusion sont relativement proches en termes de nombre d'instances résolues ou de temps d'exécution. Cela peut s'expliquer par le choix d'un nouveau cluster racine quand BTD-MAC+RST redémarre, ce qui peut être vu comme une forme simplifié de dynamicité. Par ailleurs, l'exploitation conjointe des redémarrages

Algorithme	*Min-Fill*	H_2	H_3	H_5
BTD-MAC	32 641	17 914	17 813	16 503
BTD-MAC+RST	24 456	17 514	17 050	16 235

TABLE 4 – Temps d'exécution pour BTD-MAC(+RST) selon la méthode de décomposition utilisée pour une fusion statique limitant la taille des séparateurs à 15, pour les 1 234 instances résolues par tous les algorithmes.

et des décompositions dynamiques s'avère très pertinente puisque BTD-MAC+RST+Fusion surclasse à la fois BTD-MAC+RST et BTD-MAC+Fusion. Pour terminer, nous pouvons aussi noter que BTD-MAC+RST+Fusion avec H_5 obtient les meilleurs résultats quels que soient les algorithmes de résolution et/ou de décomposition exploités.

5.4 BTD-MAC+RST+Fusion versus MAC+RST

Nous comparons à présent MAC+RST et BTD-MAC+RST+Fusion (pour $S = 50$). La figure 3(a) représente le nombre d'instances résolues pour MAC-BTD+RST+Fusion, MAC+RST et VBS (c.-à-d. le meilleur solveur virtuel parmi les deux algorithmes). D'abord, BTD-MAC+RST+Fusion résout plus d'instances que MAC+RST (1 567 instances contre 1 548). Ensuite, nous pouvons noter que le comportement de BTD-MAC+RST+Fusion est plus proche de celui de VBS que celui de MAC+RST, ce qui montre clairement la supériorité de BTD-MAC+RST+Fusion sur MAC+RST.

Nous focalisons maintenant nos observations sur les instances les plus difficiles. Parmi les 1 859 instances considérées, certaines sont résolues facilement par MAC+RST (par exemple 284 instances le sont sans aucun retour en arrière). L'exploitation de méthodes structurelles comme BTD ou ses variantes sur de telles instances n'a pas nécessairement de sens. Aussi, nous utilisons le nombre de nœuds développés par MAC+RST comme critère de difficulté. Une instance sera considérée comme difficile si ce nombre de nœuds est supérieur à $100n$ (avec n le nombre de variables). Ainsi, nous avons 577 instances considerées comme difficile. La figure 3(b) présente une comparaison des temps d'exécution de MAC-BTD+RST+Fusion et MAC+RST pour ces instances. Globalement, nous pouvons constater que, pour une bonne partie d'entre elles, MAC-BTD+RST+Fusion et MAC+RST ont un comportement similaire. En effet, pour environ 60 % des instances, l'écart entre les temps d'exécution est inférieur à 10 %. Cependant, pour les instances restantes, MAC-BTD+RST+Fusion surclasse souvent MAC+RST. Pour 16 % d'entre elles, MAC-BTD+RST+Fusion est au moins 10 fois plus rapide que MAC+RST alors que MAC+RST ne l'est que dans 1 % des cas. Enfin, l'exploitation de la structure joue ici un rôle central. En effet, 86 % des instances non résolues par MAC+RST mais résolues par BTD-MAC+RST+Fusion sont des instances structurées ayant un rapport $n/(w+1)$ supérieur à 5.

6 Conclusion

Dans ce papier, nous avons proposé deux contributions complémentaires. D'une part, nous avons présenté un nouvel algorithme pour calculer des décompositions arborescentes (à savoir H_5) permettant de borner la taille des séparateurs, qui est un paramètre crucial pour l'efficacité pratique des méthodes de résolution structurelles comme BTD. Sa complexité en temps est meilleure que celle de *Min-Fill* et il est nettement plus rapide en pratique (environ 1 000 fois plus rapide que *Min-Fill* sur un vaste ensemble d'instances). D'autre part, nous avons décrit une extension non triviale de BTD, à savoir BTD-MAC+RST+Fusion, qui peut adapter la décomposition en fusionnant dynamiquement certains clusters en fonction de la sémantique de l'instance et de connaissances acquises durant la résolution. Ainsi, notre méthode exploite des ordres sur les variables plus flexibles et peut éviter certains inconvénients que peut avoir une décomposition intiale calculée uniquement sur la base de critères structurels. En pratique, nous avons montré que BTD-MAC+RST+Fusion surclasse BTD-MAC+RST quelle que soit la méthode de décomposition exploitée. De plus, son utilisation conjointe avec H_5 conduit à l'obtention des meilleurs résultats.

Plusieurs extensions de ce travail sont possibles. D'abord, d'autres heuristiques de fusion sont envisageables en exploitant d'autres informations sémantiques. Ensuite, la rapidité de H-TD-WT ouvre des perspectives plus larges pour une modification dynamique de la décomposition en permettant des recalculs pendant la résolution et lors des redémarrages. Au-delà, des problèmes plus difficiles (optimisation, comptage ou compilation) pourraient être abordés.

Références

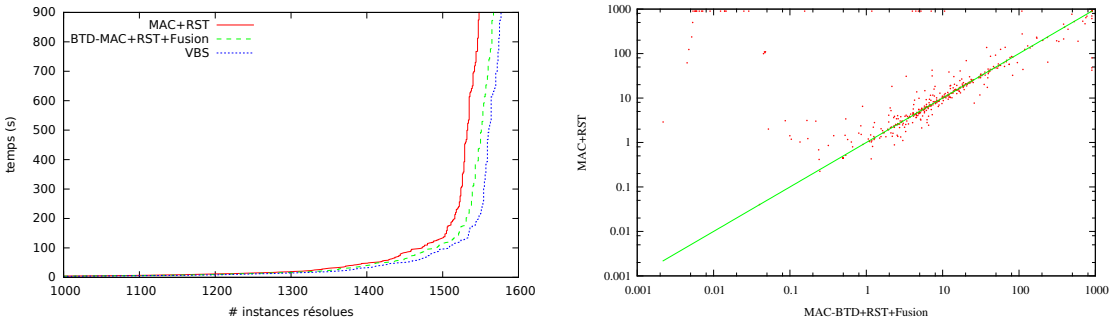
[1] D. Allouche, S. de Givry, and T. Schiex. Towards Parallel Non Serial Dynamic Programming for Solving Hard Weighted CSP. In *CP*, pages 53–60, 2010.

[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] C. Berge. *Graphs and Hypergraphs.* Elsevier, 1973.

Algorithme	*Min-Fill*		H_2		H_3		H_5	
	#rés.	temps	#rés.	temps	#rés.	temps	#rés.	temps
`BTD-MAC`	1 450	45 988	1 493	35 871	1 504	31 612	1 511	32 097
`BTD-MAC+RST`	1 537	41 722	1 549	33 328	1 553	33 164	1 564	33 145

TABLE 3 – Nombre d'instances résolues et temps d'exécution pour BTD-MAC(+RST) selon la méthode de décomposition utilisée pour une fusion statique limitant la taille des séparateurs à 15.



(a) (b)

FIGURE 3 – (a) Nombre d'instances résolues pour MAC-BTD+RST+Fusion avec H_5, MAC+RST et VBS, (b) comparaison des temps d'exécution de MAC-BTD+RST+Fusion et MAC+RST pour les 577 instances difficiles.

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

[5] C. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio Link Frequency Assignment. *Constraints*, 4 :79–89, 1999.

[6] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *AAAI*, pages 22–27, 2006.

[7] R. Dechter. *Constraint processing*. Morgan Kaufmann Publishers, 2003.

[8] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :243–282, 2000.

[9] P. Jégou, H. Kanso, and C. Terrioux. An Algorithmic Framework for Decomposing Constraint Networks. In *ICTAI*, pages 1–8, 2015.

[10] P. Jégou, S. N. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *CP*, pages 777–781, 2005.

[11] P. Jégou, S.N. Ndiaye, and C. Terrioux. Dynamic Management of Heuristics for Solving Structured CSPs. In *CP*, pages 364–378, 2007.

[12] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146 :43–75, 2003.

[13] P. Jégou and C. Terrioux. Combining Restarts, Nogoods and Decompositions for Solving CSPs. In *ECAI*, pages 465–470, 2014.

[14] P. Jégou and C. Terrioux. Tree-decompositions with connected clusters for solving constraint networks. In *CP*, pages 407–423, 2014.

[15] C. Lecoutre, C. Likitvivatanavong, S. Shannon, R. Yap, and Y. Zhang. Maintaining Arc Consistency with Multiple Residues. *Constraint Programming Letters*, 2 :3–19, 2008.

[16] C. Lecoutre, L. Saïs, S. Tabary, and V. Vidal. Recording and Minimizing Nogoods from Restarts. *JSAT*, 1(3-4) :147–167, 2007.

[17] W. Li and P. van Beek. Guiding Real-World SAT Solving with Dynamic Hypergraph Separator Decomposition. In *ICTAI*, pages 542–548, 2004.

[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] N. Eén and N. Sörensson. An Extensible SAT-solver. In *SAT*, pages 502–518, 2003.

[20] P. Refalo. Impact-based search strategies for constraint programming. In *CP*, pages 557–571, 2004.

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

[22] D. J. Rose. A graph theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.

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

Les triangles cassés, encore et encore

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

[1] IRIT, Université de Toulouse III, 31062 Toulouse, France
[2] Aix-Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296, 13397 Marseille, France
cooper@irit.fr {achref.elmouelhi, cyril.terrioux}@lsis.org

Résumé

Les triangles cassés représentent un concept important, non seulement pour la résolution en temps polynomial des problèmes de satisfaction de contraintes, mais aussi pour l'élimination de variables ou encore la réduction de la taille des domaines par la fusion des valeurs. Plus précisément, pour une variable donnée d'un CSP binaire arc-cohérent, si aucun triangle cassé ne se produit sur aucun couple de valeurs, alors cette variable peut être éliminée tout en préservant la satisfiabilité. Plus récemment, il a été démontré qu'en cas de non applicabilité de cette règle à cause de l'existence d'au moins un triangle cassé, il se peut qu'il existe deux valeurs de cette même variable sur lesquelles aucun triangle cassé ne s'est produit. Dans ce cas, ces deux valeurs peuvent être fusionnées en une seule tout en préservant la satisfiabilité.

Dans ce papier, nous montrons que sous certaines conditions, et même en cas d'existence de quelques triangles cassés sur un couple de valeurs d'une variable donnée, les deux valeurs peuvent être fusionnées sans changer la satisfiabilité de l'instance.

Abstract

Broken triangles constitute an important concept not only for solving constraint satisfaction problems in polytime, but also for variable elimination or domain reduction by merging compatible values. Specifically, for a given variable in a binary arc-consistent CSP, if no broken triangle occurs on any pair of values, then this variable can be eliminated while preserving satisfiability. More recently, it has been shown that even when this rule cannot be applied, it could be possible that for a given pair of values no broken triangle occurs. In this case, we can apply a domain-reduction operation which consists in merging these values while preserving satisfiability.

In this paper, we show that under some conditions, and even if there are some broken triangles on a pair of values of a given variable, these values can be merged without changing the satisfiability of the instance.

1 Introduction

L'étude des classes polynomiales constitue un axe de recherche important en programmation par contraintes. Récemment, une nouvelle classe hybride, appelée BTP (pour *Broken Triangle Property* [4, 5]), a été introduite. Elle possède des propriétés remarquables tant du point de vue théorique (elle capture, par exemple, plusieurs classes polynomiales existantes) que du point de vue pratique (les instances de cette classe pouvant être résolues en temps polynomial par les algorithmes MAC [18] et RFL [17] sans l'aide d'algorithmes spécifiques). Aussi, de nombreuses extensions de ce travail ont défini de nouvelles classes polynomiales [6, 7, 8, 11, 15, 16] généralisant ou étendant la classe BTP. D'autres se sont intéressées à réduire la taille des instances CSP à l'aide de versions locales de la propriété BTP. Plus précisément, étant donnée une variable d'un CSP binaire arc-cohérent [13], si aucun triangle cassé ne se produit sur aucun couple de ses valeurs, alors cette variable peut être éliminée sans modifier la satisfiabilité de l'instance [1]. Dans [2, 3], il a été démontré qu'en cas de non applicabilité de cette règle à cause de l'existence d'au moins un triangle cassé, il se peut qu'il existe deux valeurs de cette même variable sur lesquelles aucun triangle cassé ne se produit. Dans ce cas, ces deux valeurs peuvent être fusionnées en une seule tout en préservant la satisfiabilité. Cette règle de fusion, appelée BTP-fusion, est applicable pour plusieurs benchmarks de la compétition CSP. Mais, malgré cela, certains tests expérimentaux semblent indiquer qu'à l'heure actuelle, elle semble difficilement utilisable comme opération de prétraitement dans un solveur, du fait d'un temps d'exécution total trop important [2].

Au vu de ces résultats prometteurs, nous allons étudier dans ce papier une version plus légère de la fusion

par BTP tout en autorisant, sous certaines conditions, la présence de quelques triangles cassés sur le couple de valeurs à fusionner.

Dans la section suivante, nous rappelons certaines définitions et notations nécessaires pour la suite de ce travail. Dans la section 3, nous introduisons et généralisons la nouvelle règle, appelée m-wBTP-fusion, qui permet de fusionner deux valeurs d'une variable donnée même en présence de certains triangles cassés. Ensuite, nous prouvons que cette règle de fusion est maximale. Par la suite, dans la section 5, nous commençons par montrer que cette règle de fusion ne permet pas l'élimination de variables puis nous positionnons notre travail par rapport à certaines propriétés comme k-BTP [6] et WBTP [16]. Après, nous évaluons l'intérêt pratique de notre approche. Finalement, nous concluons.

2 Préliminaires

Les *problèmes de satisfaction de contraintes* (CSP, [14]) sont au cœur de nombreuses applications en intelligence artificielle et en recherche opérationnelle. Dans ce papier, nous nous intéressons uniquement aux CSP binaires qui sont formellement définis ainsi :

Définition 1 (instance binaire) *Un CSP binaire est un triplet $I = (X, D, C)$, où $X = \{x_1, \ldots, x_n\}$ est un ensemble fini de n **variables**, $D = \{D(x_1), \ldots, D(x_n)\}$ est un ensemble fini de **domaines** contenant au plus d **valeurs**, un domaine pour chaque variable et C est un ensemble de **contraintes binaires**. Chaque contrainte $C_{ij} \in C$ est un couple $(S(C_{ij}), R(C_{ij}))$ avec :*

- *$S(C_{ij}) = \{x_i, x_j\} \subseteq X$, la **portée** de la contrainte,*
- *$R(C_{ij}) \subseteq D(x_i) \times D(x_j)$, la **relation binaire** qui définit la compatibilité de valeurs.*

Si la contrainte C_{ij} n'est pas définie dans C, alors nous considérons C_{ij} comme une contrainte universelle (c'est-à-dire telle que $R(C_{ij}) = D(x_i) \times D(x_j)$).

L'interaction entre les valeurs de chaque variable à travers les relations associées aux contraintes peut être graphiquement représentée par un graphe de *microstructure* [10]. Les sommets de ce graphe sont donc les couples variable-valeur (x_i, v_i) $(v_i \in D(x_i))$ et les arêtes sont les tuples autorisés par les contraintes (c'est-à-dire qu'il existe une arête entre les sommets (x_i, v_i) et (x_j, v_j) si $(v_i, v_j) \in R(C_{ij})$). Étant donnée une instance binaire I, décider si I possède une *solution* (c'est-à-dire une *affectation* d'une valeur à chaque variable de I ne violant aucune contrainte), est un problème bien connu pour être NP-complet. Mais, en imposant quelques restrictions sur les portées et/ou les relations des contraintes, nous pouvons avoir des instances résolubles en temps polynomial. Ces instances définissent *une classe polynomiale*. La classe polynomiale BTP (pour *Broken Triangle Property)*, qui s'appuie sur le concept de *triangle cassé*, se situe au cœur des classes polynomiales puisqu'elle généralise et capture certaines classes existantes. La Broken Triangle Property impose l'absence de certains triangles cassés. Formellement, BTP est définie comme suit :

Définition 2 (Broken-Triangle Property [4, 5]) *Soient un CSP binaire I et un ordre $<$ sur les variables de I. Une paire de valeurs $v'_k, v''_k \in D(x_k)$ satisfait BTP si pour chaque couple de variables (x_i, x_j) tel que $x_i < x_j < x_k$, si*

- *$(v_i, v_j) \in R(C_{ij})$,*
- *$(v_i, v'_k) \in R(C_{ik})$ et*
- *$(v_j, v''_k) \in R(C_{jk})$,*

alors

- *soit $(v_i, v''_k) \in R(C_{ik})$,*
- *soit $(v_j, v'_k) \in R(C_{jk})$.*

Une variable x_k satisfait BTP si chaque paire de valeurs de $D(x_k)$ satisfait BTP. I satisfait BTP par rapport à l'ordre $<$ si chacune de ses variables satisfait BTP.

Cette définition peut se représenter graphiquement dans la microstructure du CSP comme indiqué dans la figure 1. Tout au long de ce papier, l'absence d'arête et les arêtes en pointillés représenteront des tuples interdits.

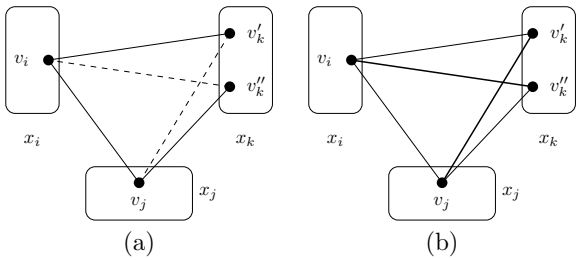


FIGURE 1 – (a) Un triangle cassé (v_i, v_j, v'_k, v''_k). (b) Un triangle non cassé (v_i, v_j, v'_k, v''_k) et satisfaisant donc la propriété BTP.

Pour la figure 1(a), ce CSP n'est pas BTP par rapport à l'ordre $x_i < x_j < x_k$ car les tuples (v_j, v'_k) et (v_i, v''_k) ne sont pas autorisés. Dans ce cas, (v_i, v_j, v'_k, v''_k) constitue un *triangle cassé* sur les valeurs v'_k et v''_k. L'existence de ce triangle cassé conduit à dire qu'il existe un triangle cassé sur x_k par rapport à x_i et x_j. Par contre, si $(v_i, v''_k) \in R(C_{ik})$ ou

$(v_j, v'_k) \in R(C_{jk})$, alors la propriété est bien vérifiée comme le montre la figure 1(b).

Nous passons maintenant à l'opération de fusion de valeurs puis nous rappelons la condition de fusion sur la base de BTP :

Définition 3 *[2]* ***Fusionner*** *les valeurs $v'_k, v''_k \in D(x_k)$ d'un CSP binaire consiste à remplacer v'_k, v''_k dans $D(x_k)$ par une nouvelle valeur v_k qui sera compatible avec toutes les valeurs qui sont compatibles avec au moins une des deux valeurs v'_k ou v''_k. Une* ***condition de fusion de valeurs*** *v'_k et v''_k d'un CSP binaire est une propriété calculable en temps polynomial de façon que lorsque cette propriété existe, alors le CSP obtenu après fusion des valeurs v'_k et v''_k est satisfiable si et seulement si le CSP de départ est satisfiable aussi.*

Le résultat suivant relie la fusion à l'absence de triangles cassés.

Proposition 1 *[2] Étant donné un CSP binaire, fusionner deux valeurs sur lesquelles aucun triangle cassé ne se produit est une condition de fusion de valeurs.*

Par exemple, dans la figure 1(b), les valeurs v'_k et v''_k sont fusionnables.

3 Les triangles légèrement cassés

L'absence de triangles cassés sur un couple de valeurs permet de les fusionner tout en préservant la satisfiabilité. Ici, nous montrons qu'il est possible de tolérer la présence de certains triangles cassés tout en conservant la possibilité de fusionner des valeurs. L'idée est inspirée d'un travail récent de Naanaa [16] sur une nouvelle extension de BTP. La propriété sera appelée m-wBTP : le paramètre m définit le nombre de variables soutenant les triangles appelés *légèrement cassés*.

3.1 1-wBTP-fusion

Nous commençons par le cas le plus basique qui s'appuie sur le nouveau concept de *triangle légèrement cassé* qui correspond à un triangle cassé soutenu par une variable.

Définition 4 *Un couple de valeurs $v'_k, v''_k \in D(x_k)$ satisfait* 1-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 au moins une variable $x_\ell \in X \setminus \{x_i, x_j, x_k\}$ tel que $\forall\ v_\ell \in D(x_\ell)$ si*

- $(v_i, v_\ell) \in R(C_{i\ell})$ *et*
- $(v_j, v_\ell) \in R(C_{j\ell})$

alors

- $(v'_k, v_\ell) \notin R(C_{k\ell})$ *et*
- $(v''_k, v_\ell) \notin R(C_{k\ell})$.

Dans ce cas, (v_i, v_j, v'_k, v''_k) sera appelé triangle légèrement cassé *soutenu par la variable x_ℓ.*

Cette définition devient plus simple quand on la représente avec le graphe de microstructure. Par exemple, dans la figure 2, on a un triangle cassé (v_i, v_j, v'_k, v''_k). Comme pour toute valeur v_ℓ de la variable x_ℓ, v_ℓ est compatible avec v_i et v_j et on a $(v'_k, v_\ell) \notin R(C_{k\ell})$ et $(v''_k, v_\ell) \notin R(C_{k\ell})$, ce triangle est un triangle légèrement cassé soutenu par x_ℓ.

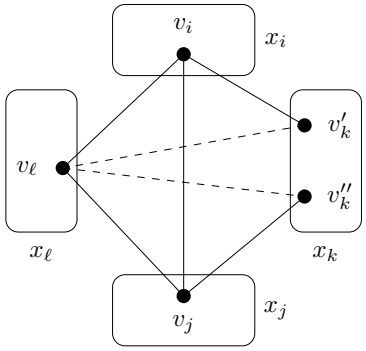


FIGURE 2 – Un triangle légèrement cassé car $(v'_k, v_\ell) \notin R(C_{k\ell})$ et $(v''_k, v_\ell) \notin R(C_{k\ell})$.

Si $(v'_k, v_\ell) \in R(C_{k\ell})$ ou $(v''_k, v_\ell) \in R(C_{k\ell})$, nous disons que $(v_\ell, v_i, v_j, v'_k, v''_k)$ forment un *triangle fortement cassé*.

La distinction de triangle légèrement cassé de triangle fortement cassé nous permet d'avoir un résultat similaire à celui de la proposition 1.

Proposition 2 *Étant donné un CSP binaire, fusionner deux valeurs $v'_k, v''_k \in D(x_k)$ qui satisfont 1-wBTP ne change pas la satisfiabilité de l'instance.*

Preuve : Soit I l'instance originale et I^f la nouvelle instance dans laquelle v'_k,v''_k ont été fusionnées en une nouvelle valeur v_k. Clairement, si I est satisfiable alors il en est de même pour I^f. Donc, il suffit de montrer que si I^f a une solution s qui affecte v_k à x_k, alors I a une solution.

Soient s', s'' deux affectations identiques à s sauf que s' affecte v'_k à x_k et s'' affecte v''_k à x_k. Supposons que ni s' ni s'' sont des solutions de I. Alors, il existe deux variables $x_i, x_j \in X \setminus \{x_k\}$ telles que $(s(x_i), v'_k) \notin R(C_{ik})$ et $(s(x_j), v''_k) \notin R(C_{jk})$. Puisque s est une solution de I^f affectant v_k à x_k, nous devons forcément avoir $(s(x_i), v''_k) \in R(C_{ik})$ et $(s(x_j), v'_k) \in R(C_{jk})$. Évidemment, nous avons $(s(x_i), s(x_j)) \in R(C_{ij})$. Dans ce cas, $(s(x_i), s(x_j), v'_k, v''_k)$ forment un triangle cassé dans I.

Par définition de 1-wBTP, il existe au moins une variable $x_\ell \in X \setminus \{x_i, x_j, x_k\}$ telle que $\forall v_\ell \in D(x_\ell)$ si

- $(s(x_i), v_\ell) \in R(C_{i\ell})$ et
- $(s(x_j), v_\ell) \in R(C_{j\ell})$

alors

- $(v'_k, v_\ell) \notin R(C_{k\ell})$ et
- $(v''_k, v_\ell) \notin R(C_{k\ell})$.

Comme $s(x_\ell)$ est compatible avec $s(x_i)$ et $s(x_j)$, elle ne peut être compatible avec ni v'_k ni v''_k. Par conséquent, $s(x_\ell)$ ne sera pas compatible avec v_k, ce qui implique que s n'est pas une solution de I^f. Or, ceci contredit notre hypothèse de départ. Donc, cette règle de fusion préserve la satisfiabilité. □

À première vue, l'existence d'un lien entre la cohérence d'arc [13] et cette définition paraît logique. En effet, imposer que les tuples (v'_k, v_ℓ) et (v''_k, v_ℓ) soient interdits peut laisser penser que le but est de rendre les valeurs v'_k et v''_k arc-incohérentes. Mais l'exemple de la figure 3 montre le contraire. En effet, bien que les deux valeurs $v'_k, v''_k \in D(x_k)$ de cette figure satisfont 1-wBTP, l'application de la cohérence d'arc ne supprime aucune valeur (et évidemment aucun tuple) puisqu'elles sont toutes arc-cohérentes. Donc, la cohérence d'arc ne supprimera pas le triangle cassé (v_i, v_j, v'_k, v''_k).

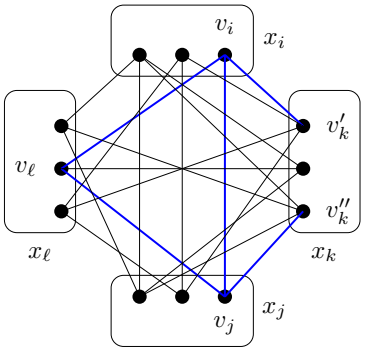


FIGURE 3 – Un CSP dont toutes les valeurs sont arc-cohérentes (en bleu le triangle légèrement cassé).

3.2 m-wBTP-fusion

Grâce à la variable x_ℓ, la fusion de valeurs sur lesquelles il existe seulement des triangles légèrement cassés ne modifie pas la satisfiabilité. En termes de microstructure, la variable x_ℓ empêche l'apparition d'une nouvelle clique de taille n (donc une solution) qui n'existait pas avant la fusion. Ce principe peut évidement être étendu à m variables ($m \leq n-3$).

Une affectation $(v_{\ell_1}, \ldots, v_{\ell_m}) \in D(x_{\ell_1}) \times \ldots \times D(x_{\ell_m})$ est une **solution partielle** si elle satisfait toutes les contraintes C_{ij} telles que $\{x_i, x_j\} \subseteq \{x_{\ell_1}, \ldots, x_{\ell_m}\}$.

Définition 5 *Un couple de valeurs $v'_k, v''_k \in D(x_k)$ satisfait m-wBTP pour $m \leq n-3$ 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 $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \subseteq X \setminus \{x_i, x_j, x_k\}$ 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 solution partielle, alors il existe $\alpha \in \{1, \ldots, r\}$ tel que $(v_{\ell_\alpha}, v'_k), (v_{\ell_\alpha}, v''_k) \notin R(C_{\ell_\alpha k})$.*

La figure 4 montre deux configurations illustrant la définition 5. Dans la première, le couple de valeurs $v'_k, v''_k \in D(x_k)$ satisfait 2-wBTP car pour l'unique solution partielle $(v_{\ell_\sigma}, v_{\ell_\gamma}, v_i, v_j)$ on a $(v_{\ell_\sigma}, v'_k), (v_{\ell_\sigma}, v''_k) \notin R(C_{\ell_\sigma k})$. Dans la deuxième, il n'existe aucune solution partielle pour l'ensemble $\{x_{\ell_\sigma}, x_{\ell_\gamma}, x_i, x_j\}$. Donc $v'_k, v''_k \in D(x_k)$ satisfait trivialement 2-wBTP.

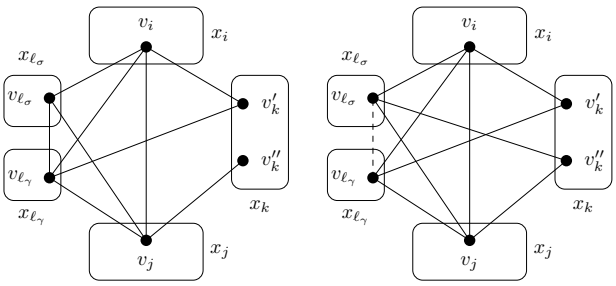


FIGURE 4 – Deux cas différents de deux valeurs v'_k et v''_k qui satisfont la 2-wBTP.

Nous généralisons maintenant le résultat de la proposition 2 sur la fusion de valeurs au cas de m variables.

Proposition 3 *Étant donné un CSP binaire, fusionner deux valeurs $v'_k, v''_k \in D(x_k)$ qui satisfont m-wBTP ne change pas la satisfiabilité de l'instance.*

Preuve : Soit I l'instance originale et I^f la nouvelle instance dans laquelle v'_k,v''_k ont été fusionnées en une nouvelle valeur v_k. Clairement, si I est satisfiable alors il en est de même pour I^f. Donc, il suffit de montrer que si I^f a une solution s qui affecte v_k à x_k, alors I a une solution.

Soient s', s'' deux affectations identiques à s sauf que s' affecte v'_k à x_k et s'' affecte v''_k à x_k. Supposons que ni s' ni s'' sont des solutions de I. Alors, il existe deux variables $x_i, x_j \in X \setminus \{x_k\}$ telles que $(s(x_i), v'_k) \notin R(C_{ik})$

et $(s(x_j), v''_k) \notin R(C_{jk})$. Puisque s est une solution de I^f affectant v_k à x_k, nous devons forcément avoir $(s(x_i), v''_k) \in R(C_{ik})$ et $(s(x_j), v'_k) \in R(C_{jk})$. On a aussi $(s(x_i), s(x_j)) \in R(C_{ij})$ puisque s est une solution de I^f. D'où, $(s(x_i), s(x_j), v'_k, v''_k)$ forme un triangle cassé dans I.

Les valeurs v'_k et v''_k satisfont m-wBTP, donc, par définition, 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 solution partielle, alors il existe $\alpha \in \{1, \ldots, r\}$ tel que $(v_{\ell_\alpha}, v'_k), (v_{\ell_\alpha}, v''_k) \notin R(C_{\ell_\alpha k})$.

Comme s est une solution pour l'instance I_f, $(s(x_{\ell_1}), \ldots, s(x_{\ell_r}), s(x_i), s(x_j))$ est nécessairement une solution partielle, donc il existe $\alpha \in \{1, \ldots, r\}$ tel que $(s(x_{\ell_\alpha}), v'_k), (s(x_{\ell_\alpha}), v''_k) \notin R(C_{\ell_\alpha k})$, ce qui implique $(s(x_{\ell_\alpha}), v_k) \notin R(C_{\ell_\alpha k})$, c'est-à-dire une contradiction car s est une solution pour l'instance I_f avec $s(x_k) = v_k$.

Par conséquent, cette règle de fusion préserve la satisfiabilité. □

La règle BTP-fusion (BTP-merging) de [2] peut être considérée comme 0-wBTP-fusion puisqu'elle s'appuie sur zéro variable de soutien pour la fusion de valeurs. La proposition suivante permet d'établir le lien entre les différentes formes de fusion sur la base de BTP.

Proposition 4 *Étant donné un CSP binaire à n variables, si un couple de valeurs $v'_k, v''_k \in D(x_k)$ satisfait m-wBTP alors il satisfait $(m+1)$-wBTP (pour $0 \leq m \leq n-4$).*

La règle BTP-fusion généralise la substitution de voisinage [9] et l'interchangeabilité virtuelle [12]. Comme m-wBTP-fusion généralise BTP-fusion pour tout $m \geq 0$, alors ceci nous conduit immédiatement au résultat suivant :

Corollaire 1 *m-wBTP-fusion généralise la substitution de voisinage et l'interchangeabilité virtuelle.*

Par ailleurs, en plus de conserver la satisfiabilité, il est possible de reconstruire en temps polynomial toutes les solutions de I à partir des solutions de toute instance I^f obtenue en appliquant à I une suite de fusions par m-wBTP-fusion. De plus, la reconstruction d'une solution de I à partir d'une solution de I^f peut s'effectuer en temps linéaire dans la taille de l'instance I. Il suffit d'appliquer les mêmes algorithmes que dans le cas de BTP-fusion [2].

4 $(n-3)$-wBTP-fusion est une condition de fusion maximale

Il est bien connu que tout couple de valeurs qui satisfait BTP peut être fusionné tout en préservant la satisfiabilité [2]. Ici, nous avons montré qu'un couple de valeurs ne satisfaisant pas BTP peut être fusionné tout en conservant la satisfiabilité du moment où il satisfait m-wBTP. En ce sens, BTP n'est pas une condition de fusion maximale. Une condition de fusion est dite *maximale* si la fusion de n'importe quel couple de valeurs ne respectant pas la propriété entraîne nécessairement la modification de la satisfiabilité de l'instance. Par contre, nous pouvons montrer que m-wBTP est une condition de fusion maximale pour $m = n-3$.

Théorème 1 *Étant donné un CSP binaire I à n variables, il n'existe aucun couple de valeurs qui ne satisfait pas m-wBTP pour $m = n-3$ et dont la fusion préserve la satisfiabilité.*

Preuve : Soient un CSP binaire I à n variables et un couple de valeurs $v'_k, v''_k \in D(x_k)$ qui ne satisfait pas m-wBTP pour $m = n-3$. Par définition de la m-wBTP-fusion, il existe un triangle cassé (v_i, v_j, v'_k, v''_k), avec $v_i \in D(x_i)$ et $v_j \in D(x_j)$, tel qu'il existe $(v_{\ell_1}, \ldots, v_{\ell_m}) \in D(x_{\ell_1}) \times \ldots \times D(x_{\ell_m})$, où $\{x_{\ell_1}, \ldots, x_{\ell_m}\} = X \setminus \{x_i, x_j, x_k\}$, tel que $(v_{\ell_1}, \ldots, v_{\ell_m}, v_i, v_j)$ est une solution partielle et pour tout $\alpha \in \{1, \ldots, m\}$ on a $(v_{\ell_\alpha}, v'_k) \in R(C_{\ell_\alpha k})$ ou $(v_{\ell_\alpha}, v''_k) \in R(C_{\ell_\alpha k})$.

Nous avons un triangle cassé, et donc :

- $(v_i, v''_k) \notin R(C_{ik})$
- $(v_j, v'_k) \notin R(C_{jk})$
- $(v_i, v'_k) \in R(C_{ik})$
- $(v_j, v''_k) \in R(C_{jk})$

Nous avons également, pour tout $\ell \in \{\ell_1, \ldots, \ell_m\}$:

- $(v_\ell, v'_k) \in R(C_{\ell k})$ ou
- $(v_\ell, v''_k) \in R(C_{\ell k})$.

Après fusion, et par définition de la fusion, la nouvelle valeur v_k satisfait $(v_\ell, v_k) \in R(C_{\ell k})$ pour tout $\ell \in \{\ell_1, \ldots, \ell_m\} \cup \{i, j\}$. Nous obtiendrons à la fin une solution formée par $v_{\ell_1}, \ldots, v_{\ell_m}, v_i, v_j$ et v_k. Donc nous venons d'introduire une solution qui n'existait pas avant puisque $(v_i, v''_k) \notin R(C_{ik})$ et $(v_j, v'_k) \notin R(C_{jk})$. Par conséquent, la fusion de n'importe quel couple de valeurs qui ne satisfait pas m-wBTP ne préserve pas la satisfiabilité. Donc, m-wBTP est une condition de fusion maximale. □

5 Fusion, élimination de variables et classes polynomiales

BTP permet la fusion de valeurs [2], l'élimination de variables [1] et la définition d'une classe polynomiale [5]. Il existe plusieurs généralisations distinctes de BTP selon la propriété étudiée. La m-wBTP est une généralisation de BTP qui permet de réduire la taille des instances via la fusion de valeurs. La m-wBTP est une condition moins restrictive que BTP et donc permet plus de fusions que BTP. La contrepartie de ce gain en nombre de fusions est le fait que la m-wBTP ne permet pas l'élimination de variables.

Dans [1], il a été démontré que, pour une variable donnée x_k d'un CSP arc-cohérent I, s'il n'existe aucun triangle cassé sur chaque couple de valeurs de $D(x_k)$, alors la variable x_k peut être éliminée de I tout en préservant la satisfiabilité. Ici, nous montrons que ce n'est pas le cas pour la m-wBTP pour $m > 0$.

Proposition 5 *Étant donnée une variable x_k d'un CSP binaire arc-cohérent I, même si chaque paire de valeurs de $D(x_k)$ satisfait m-wBTP, où $m \geq 1$, alors éliminer la variable x_k peut changer la satisfiabilité de I.*

Preuve : Soit I l'instance CSP binaire définie sur quatre variables $x_1, \ldots, x_4$ avec $D(x_i) = \{0, 1, 2\}$ $(i = 1, \ldots, 4)$ et les contraintes suivantes : $x_1 = x_2$, $x_2 = x_3$, $x_3 = x_1$, $x_1 = (x_4 + 1) \bmod 3$, $x_2 = (x_4 - 1) \bmod 3$, $x_3 = x_4$. Cette instance est arc-cohérente. Il y a trois solutions partielles $(0, 0, 0)$, $(1, 1, 1)$ et $(2, 2, 2)$ sur les variables x_1, x_2, x_3, mais I n'a pas de solution. Donc, l'élimination de la variable x_4 ne préserve pas la satisfiabilité de l'instance (voir figure 5).

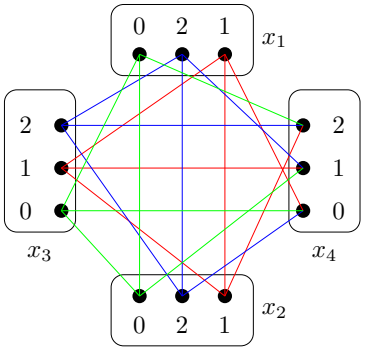


FIGURE 5 – Un CSP incohérent dont chaque couple de valeurs dans $D(x_4)$ satisfait 1-wBTP mais la suppression de x_4 entraîne l'apparition de trois solutions.

Soient x_i, x_j, x_ℓ les variables x_1, x_2, x_3 (dans n'importe quel ordre). Il existe trois triangles cassés (v_i, v_j, v'_4, v''_4) sur les variables x_i, x_j, x_4 (les triangles légèrement cassés sont représentés par des couleurs différentes dans la figure 5). Dans chacun de ces triangles cassés, nous avons $v_i = v_j$. Pour chacun de ces triangles cassés, il n'y a qu'une seule solution partielle de la forme (v_ℓ, v_i, v_j) sur les variables x_ℓ, x_i, x_j car nous avons forcément $v_\ell = v_i = v_j$. Par le choix des contraintes, les valeurs v_ℓ, v_i, v_j sont compatibles avec trois valeurs différentes dans $D(x_4)$. On peut en déduire que (v_ℓ, v'_4), $(v_\ell, v''_4) \notin R(C_{\ell 4})$ car, par la définition d'un triangle cassé, chacune des valeurs v'_4, v''_4 est compatible avec une des valeurs v_i, v_j. Par conséquent, chaque paire de valeurs $v'_4, v''_4 \in D(x_4)$ satisfait 1-wBTP.

Nous avons donné une instance I telle que chaque paire de valeurs de $D(x_4)$ satisfait 1-wBTP, mais éliminer la variable x_4 change la satisfiabilité de I. Pour les valeurs de $m > 1$, il suffit d'ajouter $m - 1$ autres variables sans contrainte à l'instance I. □

Dans l'instance I de la preuve de la proposition 5, chaque paire de valeurs dans le domaine $D(x_4)$ satisfait 1-wBTP. Cependant, après avoir effectué la fusion de deux valeurs, les deux valeurs qui restent ne satisfont pas 1-wBTP et on ne peut pas les fusionner.

Précédemment dans [6], une version allégée de BTP appelée k-BTP, qui autorise l'existence de certains triangles cassés, a été introduite. Les instances de CSP binaires qui satisfont la k-cohérence forte et k-BTP constituent une classe polynomiale. Elle est définie comme suit :

Définition 6 (k-BTP [6]) *Un CSP binaire P satisfait la propriété k-BTP pour un k donné $(2 \leq k < n)$ par rapport à un ordre $<$ sur les variables si et seulement si, pour tout sous-ensemble de 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 un couple de variables $(x_{i_j}, x_{i_{j'}})$ avec $1 \leq j < j' \leq k$ tel qu'il n'existe pas de triangle cassé sur x_{k+1} par rapport à x_{i_j} et $x_{i_{j'}}$.*

Malheureusement, et contrairement à m-wBTP, la propriété k-BTP ne peut être utilisée pour fusionner des valeurs dès que k est supérieur strictement à 2 (pour rappel 2-BTP = BTP). En effet, les valeurs v'_k et v''_k de la figure 6(a) satisfont la propriété 3-BTP puisque le triplet (x_i, x_ℓ, x_k) n'induit aucun triangle cassé sur x_k par rapport à x_i et x_ℓ. Si on fusionne v'_k et v''_k, ce CSP devient consistant alors qu'il ne l'était pas initialement. Donc, la fusion par k-BTP (pour k strictement supérieur à 2), même si elle autorise la présence de quelques triangles cassés, ne préserve pas la satisfiabilité. De plus, m-wBTP peut autoriser plus de triangles cassés que k-BTP. Par exemple, les valeurs v'_k et v''_k de la figure 6(b) satisfont 1-wBTP mais pas 3-BTP à cause des triangles cassés présents sur la va-

riable x_k quel que soit le triplet de variables incluant x_k.

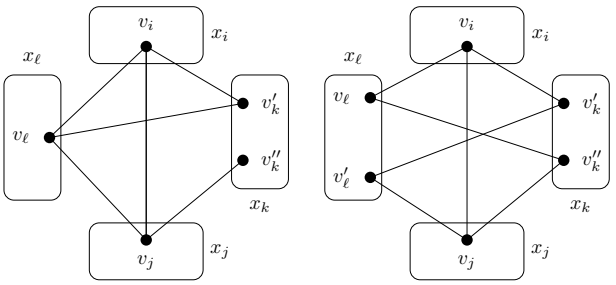


FIGURE 6 – (a) Deux valeurs v'_k et v''_k qui ne satisfont pas la 1-wBTP mais l'instance satisfait 3-BTP. (b) Deux valeurs v'_k et v''_k qui satisfont la 1-wBTP mais l'instance ne satisfait pas 3-BTP.

Naanaa a présenté deux autres généralisations de BTP qui permettent de définir des classes polynomiales [15, 16]. Il a été démontré [6] que la notion de rang directionnel $k-1$ [15] généralise strictement k-BTP. On peut en déduire que l'exemple de la figure 6(a) satisfait la propriété de rang directionnel 2, ce qui montre que la propriété de rang directionnel k (pour $k \geq 2$) ne peut être utilisée pour fusionner des valeurs (en sachant que le cas $k = 1$ correspond à BTP).

La notion de WBTP [16] a inspiré la notion de 1-wBTP, mais est différente. Nous donnons sa définition avant de démontrer que WBTP peut être vu comme une condition strictement plus forte que 1-wBTP (et donc qui permet moins de fusions).

Définition 7 (WBTP [16]) *Un CSP binaire doté d'un ordre $<$ sur ses variables satisfait la propriété WBTP (Weak Broken Triangle Property) si pour tout triplet de variables $x_i < x_j < x_k$ et pour tout $v_i \in D(x_i)$, $v_j \in D(x_j)$ tels que $(v_i, v_j) \in R(C_{ij})$, il existe une variable $x_\ell < x_k$ telle que lorsque $v_\ell \in D(x_\ell)$ est compatible avec v_i et v_j, alors nous avons $\forall v_k \in D(x_k)$,*

$$(v_\ell, v_k) \in R(C_{\ell k}) \Rightarrow \quad ((v_i, v_k) \in R(C_{ik}) \wedge (v_j, v_k) \in R(C_{jk}))$$

Proposition 6 *Si un CSP binaire doté d'un ordre $<$ sur ses variables satisfait la propriété WBTP, alors elle satisfait la propriété 1-wBTP pour chaque paire de valeurs dans le domaine de la dernière variable (selon l'ordre $<$).*

Preuve : Supposons que le CSP binaire I satisfait WBTP pour l'ordre $<$ de ses variables et soit x_k la dernière variable de I selon cet ordre. Supposons pour une contradiction, que I ne satisfait pas 1-wBTP sur une paire de valeurs $v'_k, v''_k \in D(x_k)$. Alors, par définition de 1-wBTP, il existe un triangle cassé (v_i, v_j, v'_k, v''_k) avec $v_i \in D(x_i)$, $v_j \in D(x_j)$, $v'_k, v''_k \in D(x_k)$ tel qu'il n'existe pas de variable $x_\ell \in X \setminus \{x_i, x_j, x_k\}$ tel que $\forall\ v_\ell \in D(x_\ell)$ compatible avec v_i et v_j, nous avons $(v_\ell, v'_k), (v_\ell, v''_k) \notin R(C_{\ell k})$.

Mais la propriété WBTP nous garantit qu'il existe une variable $x_\ell < x_k$ telle que $\forall v_\ell \in D(x_\ell)$ compatible avec v_i et v_j, nous avons $\forall v_k \in D(x_k)$,

$$(v_\ell, v_k) \in R(C_{\ell k}) \Rightarrow \quad ((v_i, v_k) \in R(C_{ik}) \wedge (v_j, v_k) \in R(C_{jk}))$$

L'existence du triangle cassé (v_i, v_j, v'_k, v''_k) implique que $x_\ell \notin \{x_i, x_j\}$ et donc $x_\ell \in X \setminus \{x_i, x_j, x_k\}$. D'ailleurs, puisque (v_i, v_j, v'_k, v''_k) est un triangle cassé,

$$(v_i, v_k) \in R(C_{ik}) \wedge (v_j, v_k) \in R(C_{jk})$$

est faux pour $v_k \in \{v'_k, v''_k\}$. On peut en déduire que $(v_\ell, v'_k), (v_\ell, v''_k) \notin R(C_{\ell k})$, c'est-à-dire une contradiction. □

Imposer la propriété WBTP est strictement plus fort qu'imposer 1-wBTP pour chaque paire de valeurs v'_k, v''_k dans le domaine de la dernière variable x_k. WBTP impose une condition sur toute valeur $v_k \in D(x_k)$ par rapport à la *même variable* x_ℓ, tandis que 1-wBTP (pour chaque paire de valeurs $v'_k, v''_k \in D(x_k)$) impose une condition équivalente mais pour laquelle la variable x_ℓ peut varier selon les valeurs v'_k, v''_k. L'instance de la figure 7 satisfait 1-wBTP mais ne satisfait pas WBTP car :

- seule la variable x_{ℓ_2} soutient (v_i, v_j, v'_k, v''_k),
- seule la variable x_{ℓ_1} soutient (v_i, v_j, v'_k, v'''_k).

Donc, il n'existe pas une variable qui soutient à la fois les triangles cassés (v_i, v_j, v'_k, v''_k) et (v_i, v_j, v'_k, v'''_k).

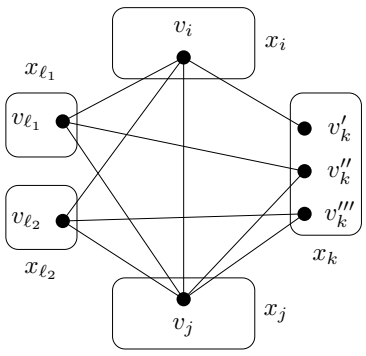


FIGURE 7 – Les couples v'_k, v''_k et v'_k, v'''_k de x_k satisfont 1-wBTP mais pas WBTP.

WBTP définit une classe polynomiale [16]. Une question qui reste ouverte est de savoir s'il est possible d'utiliser 1-wBTP pour définir une classe polynomiale qui est plus générale que celle définie par WBTP.

6 Résultats expérimentaux

Pour tester l'applicabilité de ces règles de fusion et en particulier 1-wBTP-fusion, nous avons réalisé une étude expérimentale sur l'ensemble des benchmarks binaires de la compétition internationale de solveurs CSP de 2008 [1] (soit 3 795 instances). Pour cela, l'algorithme de fusion par 1-wBTP-fusion s'inspire de celui présenté dans [3] pour la fusion par BTP-fusion. Plus précisément, étant donnée une variable x_k, nous vérifions pour chaque couple de valeurs $v'_k, v''_k \in D(x_k)$ si les deux valeurs sont fusionnables par rapport à 1-wBTP-fusion. Si un triangle cassé sur v'_k, v''_k a été trouvé, nous cherchons dans les $n-3$ variables qui restent s'il existe une variable x_ℓ qui soutient ce triangle cassé. Si on la trouve, nous continuons la recherche de triangles cassés. Sinon, le test est fini pour ces deux valeurs. Enfin, s'il n'existe pas de triangles cassés, ou s'il n'existe que des triangles légèrement cassés sur le couple v'_k, v''_k, nous les fusionnons. Nous avons implémenté ces deux algorithmes de fusion en C++ au sein de notre propre bibliothèque CSP. Les expérimentations ont été effectuées sur 8 serveurs-lames Dell PowerEdge M820 dotés de deux processeurs Intel Xeon E5-2609 v2 2,5 GHz et de 32 Go de mémoire et fonctionnant sous Linux Ubuntu 14.04.

Pour chaque benchmark, nous avons lancé BTP-fusion et 1-BTP-fusion jusqu'à la convergence vers un point fixe avec un délai de traitement d'une heure. Au total, nous avons obtenu les résultats pour 2 535 benchmarks sur 3 795 et nous avons réussi à fusionner au moins une valeur pour 1 001 instances. Dans le tableau 1, la colonne #benchmarks désigne le nombre de benchmarks pour lesquels le test a terminé en une heure. La colonne #valeurs indique la moyenne du nombre total de valeurs de ces benchmarks. Les colonnes BTP-fusion et 1-wBTP-fusion nous renseignent sur le nombre de valeurs supprimées respectivement par BTP-fusion et 1-wBTP-fusion. Quant à la figure 8, elle permet de comparer les pourcentages de valeurs supprimées par BTP-fusion et 1-wBTP-fusion instance par instance. Si, pour une majorité d'instances, les résultats sont comparables, nous pouvons remarquer que pour certaines d'entre elles, 1-wBTP-fusion fusionne significativement plus de valeurs que BTP-fusion. C'est notamment le cas pour les instances des familles `langford-*` pour lesquelles 1-wBTP-fusion fusionne de 25 à 80% des valeurs là où BTP-fusion n'en fusionne aucune.

1. Voir http ://www.cril.univ-artois.fr/CPAI08 pour plus de détails.

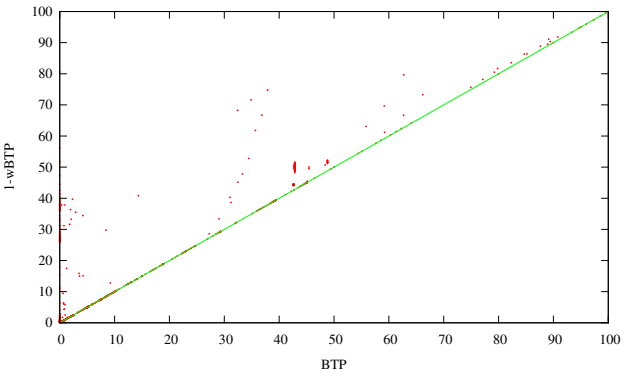


FIGURE 8 – Comparaisons des pourcentages de valeurs fusionnées par BTP et 1-wBTP.

7 Conclusion

Dans ce papier, nous avons étudié la fusion sur la base de BTP et nous avons proposé une famille de définitions qui est basée sur le concept de triangle légèrement cassé, c'est-à-dire de triangle cassé soutenu par une ou plusieurs variables pour préserver la satisfiabilité après fusion.

Il serait intéressant de vérifier si m-wBTP définit une nouvelle classe polynomiale qui généralise BTP et qui conserve ses particularités. Des éventuels liens avec la largeur arborescente devraient aussi être étudiés. Certains autres points pourraient être formellement approfondis tels que l'influence de l'ordre sur le résultat final. Par exemple, la figure 6 nous montre que si on fusionne les valeurs de x_k nous ne pouvons plus fusionner les valeurs de x_ℓ et inversement. Déterminer le meilleur ordre d'opérations de m-wBTP-fusion est NP-difficile même dans le cas $m = 0$ [2].

Références

[1] David A. Cohen, Martin C. Cooper, Guillaume Escamocher, and Stanislav Zivny. "variable elimination in binary csp via forbidden patterns". In *Proceedings of IJCAI*, 2013.

[2] Martin C. Cooper, Aymeric Duchein, Achref El Mouelhi, Guillaume Escamocher, Cyril Terrioux, and Bruno Zanuttini. Broken triangles : From value merging to a tractable class of general-arity constraint satisfaction problems. *Artificial Intelligence*, 234 :196 – 218, 2016.

Famille	#benchmarks	#valeurs	BTP-fusion	1-wBTP-fusion
BH-4-4	10	674	322	348
BH-4-7	20	2 102	883	932
ehi-85	98	2 079	891	1 045
ehi-90	100	2 205	945	1 100
graph-coloring/school	8	4 473	104	104
graph-coloring/sgb/book	26	1 887	534	534
os-taillard-4	30	2 932	1 820	1 978
rlfapScens	1	8 004	341	1 211
rlfapScensMod	6	8 788	2 415	5 169
subs	9	1 479	40	517
langford-2	22	879	0	233
langford-3	20	1 490	0	554
langford-4	16	1 784	0	504
queenAttacking	7	2 196	0	36

TABLE 1 – Résultats expérimentaux sur les benchmarks.

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

[4] Martin C. Cooper, Peter Jeavons, and Andras Salamon. Hybrid tractable CSPs which generalize tree structure. In *Proceedings of ECAI*, pages 530–534, 2008.

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

[6] Martin C. Cooper, Philippe Jégou, and Cyril Terrioux. A microstructure-based family of tractable classes for csps. In *Proceedings of CP*, pages 74–88, 2015.

[7] Achref El Mouelhi, Philippe Jégou, and Cyril Terrioux. Hidden Tractable Classes : From Theory to Practice. In *Proceedings of ICTAI*, pages 437–445, 2014.

[8] Achref El Mouelhi, Philippe Jégou, and Cyril Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. *Constraints*, 20(4) :383–413, 2015.

[9] Eugene C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI*, pages 227–233, 1991.

[10] Philippe Jégou. Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In *Proceedings of AAAI*, pages 731–736, 1993.

[11] Philippe Jégou and Cyril Terrioux. The extendable-triple property : A new CSP tractable class beyond BTP. In *Proceedings of AAAI*, pages 3746–3754, 2015.

[12] Chavalit Likitvivatanavong and Roland H. C. Yap. Many-to-many interchangeable sets of values in csps. In *Proceedings of SAC*, pages 86–91, 2013.

[13] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8 :99–118, 1977.

[14] Ugo Montanari. Networks of Constraints : Fundamental Properties and Applications to Picture Processing. *Artificial Intelligence*, 7 :95–132, 1974.

[15] Wady Naanaa. Unifying and extending hybrid tractable classes of csps. *J. Exp. Theor. Artif. Intell.*, 25(4) :407–424, 2013.

[16] Wady Naanaa. Extending the broken triangle property tractable class of binary csps. In *Proceedings of the 9th Hellenic Conference on Artificial Intelligence (SETN 2016)*, 2016.

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

[18] Daniel Sabin and Eugene C. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of ECAI*, pages 125–129, 1994.

Calcul par Contraintes de Motifs Ordonnés

Vincent Vigneron[1] *__David Lesaint__[1] **Barry Hurley**[2] **Deepak Mehta**[2] **Barry O'Sullivan**[2]

[1] LERIA, Université d'Angers, France
[2] INSIGHT Centre for Data Analytics, University College Cork, Ireland
{firstname.lastname}@{univ-angers.fr, insight-centre.org}

Résumé

La recherche de patrons ou d'itemsets fréquents a de nombreuses applications allant de la bioinformatique au marketing. Nous présentons le langage MMP (Maximal Matrix Problem) pour modéliser ces problèmes par une matrice de variables à domaines finis et un ensemble de contraintes matricielles. L'objectif est de déterminer une sous-matrice maximale cohérente, i.e., dont l'affectation des variables dans sa portée satisfait les contraintes et qui ne peut s'étendre en ligne en restant cohérente. Nous donnons une modélisation PPC du problème MMP et présentons différents types de contraintes matricielles. Nous étudions ensuite l'ordonnancement total ou partiel de patrons prélocalisés sur des séquences et permettant d'exclure des séquences prédéfinies. Nous présentons deux programmes par contraintes pour résoudre ces MMP ainsi qu'un algorithme génétique s'appuyant sur ces programmes pour passer l'échelle. Les résultats expérimentaux obtenus sur des jeux de séquences protéiques attestent de l'efficacité de l'approche.

Abstract

Itemset and pattern mining has numerous applications ranging from Marketing to Bioinformatics. We introduce a language, dubbed Maximal Matrix Problem (MMP), to model such problems. An instance of MMP is based on a matrix of finite domain variables and a set of matrix constraints. A solution is a maximal consistent submatrix whose assignment of the variables in its scope satisfies the constraints but cannot be extended over additional lines while preserving consistency. We propose a generic CP model for MMP and present various types of matrix contraints. We then tackle the problem of partially or totally ordering patterns that have been prelocalized over sequences in order to exclude predefined sequences. We present two CP models to solve these MMP together with a genetic algorithm. Experiments on datasets of protein sequences demonstrate the efficiency of the approach.

*Papier doctorant : Vincent Vigneron[1] est auteur principal.

1 Introduction

La programmation par contraintes (PPC) offre une alternative générique et efficace aux méthodes ad-hocs pour résoudre des problèmes de fouille de données. Différents modèles de PPC ont notamment été proposés pour la recherche d'itemset fréquent [3, 4] et de patrons [1, 10, 9]. Cet article propose un langage à base de contraintes, nommé Maximal Matrix Problem (MMP), pour modéliser et résoudre de tels problèmes. Une instance de MMP se définit par un domaine de matrices partiellement ordonné et un ensemble de contraintes matricielles. Une matrice résulte du choix d'une portée, prise sur un ensemble prédéfini de lignes (e.g., transactions, chaînes) et de colonnes (e.g., items, caractères), et du choix d'un coefficient pour chaque cellule de la portée, pris sur un domaine fini (e.g., absence/présence d'items, localisation de patrons).L'extension d'une matrice par adjonction de lignes ou de colonnes et choix de coefficients détermine un ordre partiel sur le domaine matriciel. L'objectif est de déterminer une matrice maximale cohérente.

Considérons la table en Figure 1 qui se compose d'un ensemble de patrons prélocalisés dans un jeu de séquences étiquetées positive ou négative. Une modélisation MMP consiste à associer les séquences aux lignes, les patrons aux colonnes et l'ensemble des localisations possibles au domaine de coefficients. Afin de discriminer séquences positives et négatives, on peut rechercher des patrons totalement ordonnés par leurs localisations sur les séquences positives mais dont l'ordre ou l'occurrence ne tient plus sur les séquences négatives. La matrice de portée $(\{s_1, s_2\}, \{\texttt{CC}, \texttt{DD}\})$ et coefficients $(s_1, \texttt{CC}) = 9$, $(s_1, \texttt{DD}) = 5$, $(s_2, \texttt{CC}) = 3$ et $(s_2, \texttt{DD}) = 1$ est solution du problème : ses coefficients sont conformes aux localisations autorisées ; elle couvre les lignes étiquetées positives ; ses colonnes satisfont l'ordre $\{\texttt{CC} > \texttt{DD}\}$; aucune combinaison de localisations autorisées pour CC et DD ne satisfait cet ordre sur s_3 ; et CC n'admet aucune localisation sur s_4.

Autrement dit, la matrice est cohérente avec les contraintes de base de données, de portée positive et d'ordre total, toute extension sur s_3 viole la contrainte d'ordre total et toute extension sur s_4 viole la contrainte de base de données. En ce sens, elle est cohérente et maximale en ligne. Elle l'est aussi en colonne, l'ordre ne pouvant être prolongé sur AA.

Nous restreignons le langage de contraintes matricielles aux contraintes de portée, de coefficients et de domaine. Les premières ne dépendent pas des choix de coefficients (e.g., un seuil de fréquence sur le nombre de lignes), les secondes ne dépendent pas des choix d'identifiants de lignes et de colonnes (e.g., égalité entre coefficients de toute portée) tandis que les contraintes de domaine restreignent les coefficients selon la portée mais indépendamment les uns des autres (e.g., compatibilité avec une base de données). Ces restrictions permettent de formuler la recherche de matrice cohérente comme un CSP et restent suffisamment souples pour modéliser des besoins pratiques en fouille d'itemsets ou de patrons. Le CSP se compose de variables représentant la portée et les coefficients de la matrice à déterminer, de contraintes d'indexation permettant d'anonymiser la portée, et d'une collection de contraintes modélisant chaque contrainte matricielle. L'encodage est linéaire en la taille maximum de matrice.

Établir la maximalité d'une matrice cohérente suppose de tester toute extension possible, c'est à dire, tout choix de coefficients sur tout sur-ensemble de lignes et de colonnes. Ce calcul est prohibitif dans le cas général puisqu'il suppose d'évaluer un nombre d'extensions doublement exponentiel. Il se simplifie si toute contrainte matricielle est monotone en ligne et en colonne (i.e., toute extension préserve la cohérence) ou bien anti-monotone (i.e., toute extension préserve l'incohérence). Sous cette hypothèse, une matrice cohérente est maximale si elle exclut chaque ligne et chaque colonne hors portée, i.e., si l'adjonction d'une ligne/colonne résulte en une matrice incohérente quel que soit le choix de coefficients des cellules adjointes. La preuve de maximalité peut ainsi se décomposer en excluant chaque ligne et chaque colonne séparément.

séquences	+/-	patrons		
		AA	CC	DD
s_1 = AACCDDAACCDD	+	{1,7}	{4,9}	{5,11}
s_2 = DDCCAADDCDD	+	{5}	{3}	{1,7,10}
s_3 = AACCAADDEDD	-	{1,5}	{3}	{7,10}
s_4 = DDAADDAA	-	{3,7}	$\emptyset$	{1,5}

FIGURE 1 – Une base de séquences constituée des positions de départ de différents patrons.

L'exclusion d'une ligne/colonne reste toutefois difficile puisque ce problème se ramène à la résolution d'un co-CSP : prouver que chaque combinaison de coefficients sur les cellules adjointes résulte en une extension violant une ou plusieurs contraintes anti-monotones. Nous proposons une résolution approchée en testant une condition d'*exclusion forte*. Il s'agit de déterminer si toutes les extensions satisfaisant les contraintes de domaine violent une même contrainte sur la ligne considérée. Chaque contrainte peut donc être traitée séparément et, en cas de succès, justifier à elle seule de l'exclusion de la ligne relativement à la base de données. Par exemple, la matrice présentée plus haut exclut fortement s_3 avec la contrainte d'ordre total et s_4 avec la base de données. A l'inverse, la matrice de portée $(\{s_1, s_2\}, \{\texttt{AA}, \texttt{DD}\})$ et de coefficients $(s_1, \texttt{AA}) = 1$, $(s_1, \texttt{DD}) = 5$, $(s_2, \texttt{AA}) = 5$, $(s_2, \texttt{DD}) = 10$ satisfait les contraintes d'ordre total $(\{\texttt{AA} < \texttt{DD}\})$ et d'écart minimum de 4 entre cellules de ligne. Elle ne s'étend pas sur s_4, les contraintes étant violées par toute extension. Pour autant, aucune des contraintes ne suffit à exclure s_4 à elle seule.

L'exclusion forte approxime donc l'exclusion totale mais elle est équivalente pour certains langages de contraintes matricielles. D'autre part, elle peut être de coût polynomial selon le langage. En outre, elle permet une algorithmique modulaire qui combine des propagateurs PPC dédiés à la cohérence avec des routines d'exclusion forte propres à chaque contrainte et appliquées à chaque ligne/colonne hors portée (routines appelées *co-propagateurs*). Enfin, les contraintes constituent des explications qui peuvent être mise à profit dans un contexte interactif où l'utilisateur affine son problème par ajout et retrait de contraintes, ou ré-étiquetage de lignes ou colonnes.

Le reste de l'article s'organise comme suit. La section 2 définit le problème MMP et donne une formulation PPC du calcul de matrice cohérente. La section 3 esquisse une modélisation MMP des problèmes de recherche d'itemsets et de patrons et présente quelques contraintes matricielles fondamentales. La section 4 se focalise sur la recherche de matrice de dimension bornée et sujette aux contraintes de base de données et de préservation de l'ordre requis, qu'il soit total ou maximum partiel. Nous montrons que ces problèmes sont NP-complet. Sont ensuite présentés des co-propagateurs pour les contraintes d'ordre total et d'ordre partiel maximum ainsi qu'un modèle complet et une approche génétique pour résoudre chaque problème. La section 5 présente une validation expérimentale sur des jeux de données biologiques qui confirment l'intérêt et les performances de l'approche. Enfin, la section 6 fait un rapide état de l'art et la section 7 présente quelques perspectives.

2 Maximal Matrix Problem

Nous formalisons dans cette section les notions de matrice, contrainte et extension sur un type matriciel et présentons le problème de décision MMP. On notera $[k]$ l'intervalle $\{j \in \mathbb{N} \mid 1 \leq j \leq k\}$ pour tout $k \in \mathbb{N}^+$, et $f(S)$ l'ensemble $\{f(u) \mid u \in S\}$ pour tout $f : A \to B$, $S \subseteq A$. Un type matriciel se définit par un ensemble ordonné d'identifiants de lignes et de colonnes et par un domaine fini pour les coefficients matriciels. L'ordre choisi est arbitraire et sans incidence sur les solutions d'une ins-

$$var\ 1..p : m; \quad (1)$$
$$array[1..p]\ of\ var\ 1..p : s1; \quad (2)$$
$$nvalue(s1, m); \quad (3)$$
$$increasing(s1); \quad (4)$$
$$s1[p] == s1[m]; \quad (5)$$
$$var\ 1..q : n; \quad (6)$$
$$array[1..q]\ of\ var\ 1..q : s2; \quad (7)$$
$$nvalue(s2, n); \quad (8)$$
$$increasing(s2); \quad (9)$$
$$s2[q] == s2[n]; \quad (10)$$
$$array[1..p, 1..q]\ of\ var\ D : v; \quad (11)$$

FIGURE 2 – Modélisation PPC de type matriciel.

$$among(p, s1, positives); \quad (1)$$
$$n \geq threshold; \quad (2)$$
$$\forall i = 1..p, j = 1..q,$$
$$(m == i/\backslash n == j) => allequal([v[k, l]|k = 1..i, l = 1..j]); \quad (3)$$
$$\forall i = 1..p, j = 1..q,\ v[i, j]\ in\ db[s1[i], s2[j]]; \quad (4)$$

FIGURE 3 – Modélisation PPC de contraintes matricielles.

tance de MMP. Sans perte de généralité, on supposera donc que les lignes et les colonnes sont numérotées (i.e., identifiées par des entiers) et triées en ordre croissant.

Définition 1 (Type matriciel)**.** *Un type matriciel M est un triplet $\langle p, q, D\rangle$ tel que $p \in \mathbb{N}^+$, $q \in \mathbb{N}^+$, et D est un ensemble fini. Les éléments de $[p]$, $[q]$ et D sont appelés, respectivement, lignes, colonnes, et coefficients de M.*

Une matrice se définit par le choix d'une portée et de coefficients sur cette portée relativement à un type donné. Ces choix caractérisent la dimension de la matrice en termes de nombre de lignes et de colonnes. Portée et coefficients se définissent par des fonctions d'indexation qui partagent les mêmes domaines de définition et assurent une représentation canonique de toute matrice.

Définition 2 (Matrice)**.** *Soient $M = \langle p, q, D\rangle$ un type matriciel, $m \in [p]$, et $n \in [q]$. Une matrice de type M et dimension (m, n) est une paire (s, v) telle que $s = (s_1, s_2)$, $s_1 : [m] \to [p]$, $s_2 : [n] \to [q]$, $v : [m] \times [n] \to D$ avec s_1 et s_2 strictement croissantes. s et v sont appelés portée et coefficients de x, respectivement.*

On notera $(s, v)_{m,n}$ pour indiquer la dimension (m, n) d'une matrice (s, v). L'ensemble des matrices de type M et taille (m, n) est dénoté $[\![M_{m,n}]\!]$ et l'ensemble des matrices de type M est dénoté $[\![M]\!] = \bigcup_{m\in[p],n\in[q]} [\![M_{m,n}]\!]$. La figure 2 présente une modélisation PPC de type matriciel. Les variables m (1) et n (6) représentent le nombre de lignes et de colonnes de la matrice à déterminer. Les tableaux de variables d'indices $s1$ (2) et $s2$ (7) représentent les fonctions d'indexation de portée. Les contraintes *nvalue* (3), *increasing* (4) et *élément* (5) assurent que $s1$ est strictement croissante sur $[m]$, les valeurs d'indice de ligne au delà du rang m étant forcées à celle du rang m. La modélisation est symétrique pour les colonnes (contraintes (8), (9) (10)). Le tableau de variables v (11) modélise les coefficients à déterminer, comme expliqué ci-après.

Une contrainte matricielle est une relation unaire sur le domaine matriciel qui se décompose en $p \times q$ relations. Chaque sous-relation est associée à une dimension de matrice (m, n) et utilisée de manière exclusive pour tester toute matrice (m, n) (condition (1) de la définition 3). Nous nous limitons à trois classes de contraintes : les contraintes de portée (condition (2)), les contraintes de coefficients (condition (3)), et les contraintes de domaine (condition (4)). La sémantique des contraintes de portée est indépendante du choix de coefficients. Celle des contraintes de coefficients est indépendante du choix de portée. Autrement dit, ces contraintes s'appliquent à des portées anonymisées. Les contraintes de domaine sont les seules contraintes matricielles liant portée et coefficients mais sont décomposables par restriction au niveau cellulaire.

Nous formalisons l'opération de restriction comme suit. Pour toute matrice $x = (s, v)_{m,n}$ et tout $I \subseteq s_1([m])$, $J \subseteq s_2([n])$, il existe $K = \{i_1, \ldots i_{|I|}\} \subseteq [m]$ et $L = \{j_1, \ldots j_{|J|}\} \subseteq [n]$ tels que $i_1 < i_2 < \ldots i_{|I|}$, $j_1 < j_2 < \cdots j_{|J|}$, $s_1(K) = I$ et $s_2(L) = J$. K et L sont uniques et on note $s_{1I} : [|I|] \to [p]$, $s_{2J} : [|J|] \to [q]$, $v_{IJ} : [|I|] \times [|J|] \to D$ définis par $s_{1I}(k) = s_1(i_k)$, $s_{2J}(l) = s_2(j_l)$ et $v_{IJ}(k, l) = v(i_k, j_l)$ ($k \in [|I|]$, $l \in [|J|]$). $((s_{1I}, s_{2J}), v_{IJ})$ est la matrice, dénotée x_{IJ}, obtenue par restriction de x aux lignes de I et colonnes de J. On notera x_{ij} la matrice $x_{\{i\}\{j\}}$ pour tout $i \in s_1([m])$, $j \in s_2([n])$.

Définition 3 (Contrainte matricielle)**.** *Soit M un type matriciel. Une contrainte c de type M est une relation unaire sur $[\![M]\!]$ associée à un ensemble $\{c_{m,n} | \ c_{m,n} \subseteq [\![M_{mn}]\!], m \in [p], n \in [q]\}$ et vérifiant la condition*

$$\forall m \in [p], n \in [q], x \in [\![M_{m,n}]\!], c(x) \Leftrightarrow c_{m,n}(x) \quad (1)$$

et l'une au moins des conditions suivantes :

$$\forall (s, v), (s, v') \in [\![M]\!], c((s, v)) \Leftrightarrow c((s, v')) \quad (2)$$
$$\forall (s, v), (s', v) \in [\![M]\!], c((s, v)) \Leftrightarrow c((s', v)) \quad (3)$$
$$\forall x = (s, v)_{m,n} \in [\![M]\!], c(x) \Leftrightarrow \bigwedge_{\substack{i\in s_1([m])\\ j\in s_2([n])}} c(x_{ij}) \quad (4)$$

La figure 3 illustre ces trois classes de contraintes. La contrainte de portée (1) impose que les lignes soient étiquetées positive par le biais de la contrainte $among$. La contrainte de portée (2) impose un seuil de fréquence en colonne. La contrainte de coefficients (3) impose l'égalité de tout coefficient dans la portée par le conditionnement de $p \times q$ contraintes $allequal$. Enfin, la contrainte de domaine (4) impose la compatibilité avec une base de données db encodée par un tableau $p \times q$ de sous-domaines de D (qui peuvent être vides ou égaux à D).

L'extension en ligne d'une matrice traduit l'adjonction d'une seule ligne à la portée avec choix de coefficients,

mais sans ajout de colonnes ni changement des coefficients de la matrice. L'extension détermine donc un ordre partiel sur le type matriciel.

Définition 4 (Extension matricielle)**.** *Soient* $x = (s, v)_{m,n}$ *et* $x' = (s', v')_{m',n'}$ *deux matrices de type* $\langle p, q, D\rangle$ *et* $i \in [p]$*.* x' *étend* x *sur* i*, dénoté* $x \sqsubset_i x'$*, si et seulement si* $\{i\} = s'_1([m']) \setminus s_1([m]) \wedge s'_2 = s_2 \wedge v'_{s_1([m])s_2([n])} = v$*.*

Une instance de MMP se définit par un type matriciel et un ensemble de contraintes scindé en contraintes de domaine, de coefficients et de portée. Comme leur nom l'indique, ces dernières sont utilisées pour controler la portée de matrice et ne sont donc pas prises en compte dans le calcul de maximalité, i.e., l'exclusion ne se jauge qu'avec les contraintes de domaine et de coefficients. Précisément, une solution de MMP est une matrice cohérente avec les contraintes et maximale pour l'exclusion forte relativement aux contraintes de domaine et de coefficients, i.e., une matrice qui satisfait les contraintes sur sa portée et dont l'extension en ligne viole systématiquement une même contrainte, la contrainte étant à déterminer pour chaque ligne. Étant donné un ensemble C de contraintes de type M et $x \in [\![M]\!]$, on utilisera $C(x)$ en lieu et place de $\bigwedge_{c\in C} c(x)$.

Définition 5 (MMP)**.** *Une instance de MMP est une paire* $\mu = \langle M, C\rangle$ *tel que* $M = \langle p, q, D\rangle$ *est un type matriciel et* $C = C_d \uplus C_s \uplus C_v$ *est un ensemble de contraintes de type* M *partitionné en contraintes de domaine* C_d*, de portée* C_s*, et de coefficients* C_v*.* μ *est satisfiable si et seulement si la proposition suivante est vérifiée :*

$$\exists x \in [\![M]\!], C(x) \wedge \bigwedge_{i\in[p]} \bigvee_{c\in C_v} \Big(\forall y \in [\![M]\!], (C_d(y) \wedge x \sqsubset_i y) \Rightarrow \neg c(y)\Big).$$

Une solution x de $\langle M, C\rangle$ est donc une matrice cohérente (i.e., satisfaisant C) et maximale (i.e. excluant fortement chaque ligne i hors portée par une contrainte de coefficients c spécifique à i). La figure 4 présente une modélisation PPC de la procédure d'exclusion forte dans le cas où, pour chaque contrainte de coefficients de l'instance, ce problème est polynomial. La procédure repose sur un un tableau de variables pseudo-booléennes r (1) où $r[i]$ vaut 1 si et seulement si la ligne i est dans la portée (contraintes (2) et (3)). Chacune de ces variables conditionne la tentative d'exclusion forte de la ligne si elle est hors portée. L'exclusion s'établit par une disjonction de contraintes d'exclusion (4). La contrainte d'exclusion $cop_k(i, db, s1, s2, v)$ est associée à la k-ième contrainte de coefficients, c, et a pour sémantique $(\forall y \in [\![M]\!], (d_B(y) \wedge x \sqsubset_i y) \Rightarrow \neg c(y))$, où x correspond à la matrice candidate $(m, n, s1, s2, v)$. On présente en section4 des co-propagateurs décidant ces contraintes pour les contraintes matricielles d'ordre.

$$array[1..p]\ of\ var\ 0..1 : r; \quad (1)$$
$$m = \sum_{i=1..p} r[i]; \quad (2)$$
$$\forall i = 1..p, r[s1[i]] == 1; \quad (3)$$
$$\forall i = 1..p, (r[i] == 0) => (cop_1(i, db, s1, s2, v)||..||cop_t(i, db, s1, s2, v)); \quad (4)$$

FIGURE 4 – Modélisation PPC de l'exclusion forte.

3 Modélisation et contraintes MMP

Nous présentons dans cette section quelques prédicats matriciels ainsi qu'une modélisation possible de problèmes de recherche d'itemsets et de patrons. La table 1 formalise les prédicats de base de données, de couverture, de cardinalité, de fréquence, d'ordre total et d'ordre partiel maximum. Chaque prédicat y est défini par sa signature (symbole et paramètres), sa sémantique et la sémantique du problème d'exclusion forte qui lui est associé. Le prédicat de base de données $\mathtt{d_B}$ est paramétré par une base de données B qui se définit par la matrice $p \times q$ des coefficients autorisés pour chaque cellule. Ce prédicat est anti-monotone pour l'extension $\sqsubset_i$, i.e., toute extension de matrice violant $\mathtt{d_B}$ viole également $\mathtt{d_B}$. Le prédicat de couverture $\mathtt{cover_R}$ est paramétré par un ensemble de lignes R et satisfait par toute matrice dont les lignes incluent R. Les prédicats de cardinalité $\mathtt{card}_{\leq \mathtt{w}}$ et de fréquence $\mathtt{freq}_{\leq \mathtt{w}}$ imposent qu'une matrice ait moins de w colonnes et w lignes, respectivement. Ces trois prédicats sont des prédicats de portée, indépendants des choix de coefficients.

Le prédicat $\mathtt{total}_<$ est paramétré par un ordre total strict $<$ sur D. $<$ induit sur chaque ligne de matrice un ordre partiel entre les colonnes qui dépend de la valeur des coefficients. $\mathtt{total}_<$ impose que ces ordres soient totaux et égaux sur toutes les lignes de la portée. Ce prédicat est clairement anti-monotone pour l'extension et exclure une ligne sur la base d'une matrice le satisfaisant revient à prouver l'insatisfiabilité d'un CSP binaire qui ne comporte que des contraintes d'inégalité $<$ (cf. table 1). Cette classe de CSP est polynomiale et nous présentons en section suivante un co-propagateur sous forme de CSP.

$\mathtt{partial}_{<,\alpha}$ est similaire à $\mathtt{total}_<$ mais se borne à calculer l'ordre partiel maximum qui est induit par $<$ entre les colonnes et commun à l'ensemble des lignes, ordre dénoté α en table 1. Cet ordre existe et est unique pour toute matrice donc $\mathtt{partial}_{<,\alpha}$ est toujours vérifiée. Cependant, nous utilisons ce prédicat pour l'exclusion en imposant qu'aucune extension de matrice cohérente ne préserve l'ordre partiel maximum qui lui est associé. Le problème d'exclusion reste polynomial et revient à montrer l'incohérence du CSP défini en table 1. Nous en donnons un co-propagateur en section suivante.

Nous présentons quelques modélisations MMP de problèmes de fouille utilisant ces prédicats. D'une part, la re-

TABLE 1 – Exemples de prédicats matriciels sur $M = \langle p, q, D\rangle$. Chaque ligne correspond à un prédicat c dont on donne la sémantique $c(x)$ pour $x = (s, v) \in [\![M]\!]$, et celle du problème d'exclusion d'une ligne i.

Prédicat	**Paramètres**	$c(x) \Leftrightarrow$	x exclut i avec c ssi
d_B	$B : [p] \times [q] \to 2^D$	$\bigwedge_{i\in[m],j\in[n]}(v(i,j) \in B(s_1(i), s_2(j)))$	$\bot$
$\texttt{cover}_\texttt{R}$	$R \subseteq [p]$	$R \subseteq s_1([m])$	N/A
$\texttt{card}_{\leq\texttt{w}}$	$w \in [q]$	$w \geq n$	N/A
$\texttt{freq}_{\leq\texttt{w}}$	$w \in [p]$	$w \leq m$	N/A
$\texttt{total}_<$	$<$ ordre total strict sur D	$\bigwedge_{j,k\in[n],j\neq k}$ $((\bigwedge_{i\in[m]}(v(i,j) < v(i,k)))$ $\vee(\bigwedge_{i\in[m]}(v(i,j) > v(i,k))))$	CSP (X, D, C) insatisfaisable avec $X = \{X_1, \ldots, X_n\}$ $X_j \in D_j,\ D_j = B(i, s_2(j))\ (j \in [n])$ $C = \{(X_j < X_k) \mid j, k \in [n]\,,\ v_{s_1(1)s_2(j)} < v_{s_1(1)s_2(k)}\}$
$\texttt{partial}_{<,\alpha}$	$<$ ordre total strict sur D	$\alpha : [n] \times [n] \to \mathbb{B}$ $\alpha(j,k) = \bigwedge_{i\in[m]}(v(i,j) < v(i,k))$	CSP (X, D, C) insatisfaisable avec $X = \{X_1, \ldots, X_n\}$ $X_j \in D_j,\ D_j = B(i, s_2(j))\ (j \in [n])$ $C = \{(X_j < X_k) \mid j, k \in [n]\,, \alpha(j,k)\}$

cherche d'itemsets fréquents parmi p items et q transactions (où chaque transaction est un sous-ensemble d'items) se modélise par le type matriciel $\langle p, q, \{1\}\rangle$ et une base de données B vérifiant $B(i, j) = D$ si l'item j appartient à la transaction i. Autrement dit, $B(i, j) = \emptyset$ si l'item j n'appartient pas à la transaction i. On garantit ainsi que seuls les itemsets présents dans une transaction satisfont la contrainte de domaine. On peut ensuite rechercher des itemsets fréquents en paramétrant le prédicat $\texttt{freq}_{\geq\texttt{w}}$.

Un ensemble de p chaînes de caractères sur lesquelles q patrons ont été prélocalisés (en calculant toutes les positions de départ possibles) peut se modéliser par un type matriciel $\langle p, q, D\rangle$ à domaine entier et une base de données consignant les localisations admissibles de chaque patron dans chaque séquence. On peut ainsi rechercher un ensemble commun de patrons et le contraindre par des prédicats de fréquence, cardinalité, et ordre partiel ou total. Dans le cas où les patrons correspondent à des sous-chaînes ou sous-séquences à trous constants, une matrice satisfiant le prédicat d'ordre total permet un alignement des chaînes couvertes. Lorsque les patrons correspondent aux lettres de l'alphabet utilisé, le prédicat d'ordre total détermine des sous-séquences communes sans répétition. L'usage supplémentaire d'un prédicat d'écart nul (non présenté ici) permet de calculer des sous-chaînes communes sans répétition. Enfin, l'association à chaque lettre d'un nombre de colonnes correspondant au nombre maximum d'occurrences de la lettre dans une chaîne permet de rechercher sous-chaîne et sous-séquence commune au sens général.

4 Calcul de matrices ordonnées

Dans cette section, nous abordons les prédicats d'ordre total et d'ordre partiel maximum. On étudie les problèmes dont toute solution est une matrice compatible avec une base de données (prédicat $\texttt{d}_\texttt{B}$), dont les lignes recouvrent un ensemble R prédéfini (prédicat $\texttt{cover}_\texttt{R}$) et dont les colonnes sont ordonnées, totalement (prédicat $\texttt{total}_<$) ou partiellement (prédicat $\texttt{partial}_{<,\alpha}$). Une matrice solution exclut donc toute ligne hors portée, soit par incompatibilité avec la base de données, soit par impossibilité de préserver l'ordre induit. On se restreint à rechercher parmi l'ensemble de solutions celles dont la fréquence et la cardinalité sont bornées supérieurement. Le problème MMP résultant est NP-complet dans ses variantes avec ordre total, avec ordre partiel maximum, ou sans contrainte d'ordre. Soit $\Gamma = \langle \Gamma_d, \Gamma_s, \Gamma_v\rangle$ où Γ_d, Γ_s et Γ_v sont des ensembles de prédicats de domaine, de portée, et de coefficients, respectivement. On note MMP$\langle\Gamma\rangle$ la classe d'instances MMP dont les contraintes de domaine, de portée et de coefficients instancient respectivement les prédicats de Γ_d, Γ_s et Γ_v sur le type matriciel choisi.

Théorème 1. *MMP$\langle\Gamma\rangle$ est NP-complet si* $\Gamma_d = \{\texttt{d}_\texttt{B}\}$, $\Gamma_s = \{\texttt{freq}_{\leq\texttt{k}}, \texttt{card}_{\leq 1}\}$ *et* $\Gamma_v \subseteq \{\texttt{total}_<, \texttt{partial}_{<,\alpha}\}$.

Proof. (1) On considère d'abord la classe d'instances sans contraintes d'ordre, i.e., $\Gamma_v = \emptyset$. Vérifier qu'une matrice est solution revient d'abord à tester si elle satisfait les contraintes de base de données, fréquence ($\leq k$) et cardinalité ($\leq l$). Ces tests s'effectuent clairement en temps polynomial. Vérifier qu'une matrice cohérente est maximale revient à tester l'exclusion de chaque ligne hors portée avec la base de données. Dans ce cas, il suffit de tester pour chaque ligne s'il existe une cellule incompatible avec la base pour pouvoir l'exclure. La vérification de solution est donc polynomiale et le problème est dans NP. Pour établir la NP-complétude, on réduit le problème de couverture par ensembles. Etant donné un entier l, un ensemble U fini, et un sous-ensemble S de l'ensemble des parties de U, il s'agit de déterminer l'existence d'un sous-ensemble T de S, de taille inférieure à l, et tel que l'union

des éléments présents dans les sous-ensembles de T est égal à U. La réduction consiste à définir le type matriciel $\langle k + |U|, |S|, D\rangle$, D quelconque, où l'on associe la colonne j à chaque élément s_j de S, la ligne i à chaque élément u_i de U ainsi que $k > 0$ lignes supplémentaires. La base de données autorise toute valeur pour toutes les cellules de ces k lignes. Elles sont donc nécessairement couvertes par toute matrice solution. Pour toute cellule (i, j) ($u_i \in U, s_j \in S$), on pose $B(i, j) = \emptyset$ ssi $u_i \in s_j$. On pose enfin les contraintes $\texttt{freq}_{\leq \texttt{k}}$ et $\texttt{card}_{\leq 1}$. Ainsi, une matrice maximale doit exclure chaque ligne u_i, i.e., une de ses colonnes correspond à un élément de S qui contient u_i. L'ensemble des couvertures solutions de taille $\leq l$ est donc en bijection avec l'ensemble des matrices solutions de ce MMP et la réduction s'effectue en temps polynomial. Le problème MMP$\langle\{\texttt{d}_\texttt{B}\}, \{\texttt{freq}_{\leq \texttt{k}}, \texttt{card}_{\leq 1}\}, \{\}\rangle$ est donc NP-complet. (2) On considère la classe d'instances avec contrainte d'ordre total, i.e., $\Gamma_v = \{\texttt{total}_<\}$. La vérification de matrice solution reste polynomiale car la cohérence d'une matrice avec la contrainte d'ordre total consiste à comparer des couples de coefficients et l'exclusion avec cette contrainte est aussi polynomiale (cf. table 1). Pour établir la NP-complétude, on considère la classe d'instances $\mu = \langle\langle p, q, D\rangle, C\rangle$ construites pour $\Gamma_v = \emptyset$ et vérifiant $D = [q]$ et $B(i, j) \subseteq \{j\}$ ($i \in [p], j \in [q]$). Cette classe définit un problème NP-complet selon la preuve précédente. De plus, toute matrice cohérente avec la base de données satisfait nécessairement l'ordre total induit entre colonnes par la base, et cet ordre ne peut donc exclure aucune ligne hors portée. Cette classe se réduit donc à la classe d'instances avec contrainte d'ordre total qui est donc NP-complet. (3) Une preuve similaire s'établit pour la classe avec contrainte d'ordre partiel en utilisant cette fois un domaine singleton qui n'autorise donc que des ordres partiels maximaux vides. □

Nous présentons deux modèles PPC pour les classes de MMP avec contrainte d'ordre total et avec contrainte d'ordre partiel maximum. On se place dans le cadre de la recherche d'ensemble ordonné de patrons qui ont été prélocalisés sur un jeu de séquences étiquetées positive ou négative. La figure 7 présente deux exemples de solutions pour deux instances du problème d'ordre partiel. On suit la modélisation donnée en section 3 où les séquences, patrons et localisations sont assimilés, respectivement, aux lignes, colonnes et coefficients. En théorie, on cherche à calculer des matrices maximales et de portée bornée supérieurement en fréquence et cardinalité. Plutôt que d'explorer le front Pareto des solutions correspondant à cet objectif bi-critère, on recherche en priorité les solutions de fréquence minimum et, parmi ces solutions, celles de plus petite cardinalité. Puisque les séquences positives doivent être couvertes, on cherche donc à exclure le maximum de séquences négatives avec un nombre minimal de patrons.

Dans les deux modèles présentés, p et q désignent respectivement le nombre de séquences et le nombre de patrons. $\mathcal{P}^+$ et $\mathcal{P}^-$ sont deux sous-ensembles disjoints de $[p]$ représentant l'ensemble des séquences positives et l'ensemble des séquences négatives. $L[i, j]$ correspond à l'ensemble des localisations du patron j dans la séquence i. $L.row(i)$ correspond aux ensembles des localisations des patrons dans la séquence i.

$$\bigvee_{j\in[q]} c[j] \quad (1)$$

$$\forall i \in [p] : \quad sort\_perm(l.row(i), o, sl.row(s)) \quad (2)$$

$$\forall j \in [q] : \quad c[j] \leftrightarrow (o[j] \leq card) \quad (3)$$

$$\forall i \in [p] : \quad element(card, sl.row(i), last[i]) \quad (4)$$

$$\forall i \in \mathcal{P}^+ : \quad last[i] \notin \Gamma \quad (5)$$

$$\forall i \in \mathcal{P}^- \forall j \in [q] : \quad element(o[j], L\Gamma.row(i), sL\Gamma[i, j]) \quad (6)$$

$$\forall i \in \mathcal{P}^- : \quad minimum(sl[1], sL\Gamma[i, 1]) \quad (7)$$

$$\forall i \in \mathcal{P}^- \forall j \in [2..q] : \quad min\_at\_least(sl[j], sL\Gamma[i, j], sl[j-1]) \quad (8)$$

$$\forall i \in \mathcal{P}^- : \quad r[i] \leftrightarrow (last[i] \notin \Gamma) \quad (9)$$

$$minimize \quad q.\sum_{i\in\mathcal{P}^-} r[i] + card \quad (10)$$

FIGURE 5 – Modèle pour l'ordre total.

4.1 Un modèle PPC pour l'ordre total

La figure 5 présente le modèle PPC pour la résolution lexicographique du problème MMP$\langle\{\texttt{d}_\texttt{B}\}, \{\texttt{cover}_{\mathcal{P}^+}, \texttt{freq}_{\leq \texttt{k}}, \texttt{card}_{\leq 1}\}, \{\texttt{total}_<\}\rangle$.

Notations. $\epsilon = max_{i\in[p]}(|s_i|) + 1$ est une valeur strictement supérieure à l'ensemble des localisations possibles. $\Gamma = \{\epsilon, \ldots, \epsilon + [q]\}$ est un ensemble de valeurs qui permet de reproduire sur les séquences négatives tout ordre total commun aux séquences positives. Pour une séquence négative, avoir recours à des valeurs de Γ pour reproduire un ordre total signifie que cet ordre ne peut être reproduit avec les localisations autorisées par la base de données, donc la séquence ne supporte pas l'ordre total.

Variables. Pour tout $j \in [q]$, $c[j]$ est une variable booléenne qui dénote l'inclusion du patron j dans la solution. l est une matrice de variables entières telle que chaque cellule $l[i, j] \in \Gamma \cup L[i, j]$ correspond à la localisation choisie pour le patron j dans la séquence i. $o[j]$ est une variable entière qui correspond au rang du patron j dans l'ordre total. Pour tout $i \in \mathcal{P}^-$, $r[i]$ est une variable booléenne qui modélise l'exclusion de la séquence i, c'est-à-dire si l'ordre total ne peut pas être reproduit sur i. sl est une matrice d'entiers telle que chaque ligne $sl.row(i)$ correspond à la ligne $l.row(i)$ triée par ordre croissant. sL est une matrice de variables d'ensembles d'entiers telle que chaque ligne $sL.row(i)$ correspond à la ligne $L.row(i)$ permutée avec la fonction définie par o. $card$ est une variable entière correspondant au nombre de patrons sélectionnés. Enfin, $last[i]$ est une variable entière correspondant à la localisation du dernier patron de l'ordre total dans la séquence i.

Contraintes. L'équation (1) impose la sélection d'au

$$\bigvee_{j\in[q]} c[j] \quad (1)$$

$$\forall j, j' \in [q]: \quad a[j,j'] \leftrightarrow (c[j] \wedge c[j'] \wedge \bigwedge_{i\in\mathcal{P}^+} (l[i,j] < l[i,j'])) \quad (2)$$

$$\forall i \in \mathcal{P}^-: \quad check\_partial\_order(c, a, L.row(i), r[i]) \quad (3)$$

$$minimise \quad q.\sum_{i\in\mathcal{P}^-} r[i] + \sum_{j\in[q]} c[j] \quad (4)$$

FIGURE 6 – Modèle pour l'ordre partiel.

moins un patron. L'équation (2) ordonne chaque séquence positive en utilisant la même permutation. [1] L'équation (3) impose que les rangs des patrons sélectionnés doivent précéder les rangs des autres patrons. L'équation (4) enregistre pour chaque séquence la localisation du dernier patron de l'ordre. L'équation (5) impose pour chaque séquence positive que la localisation du dernier patron sélectionné doit être valide, c'est-à-dire en dehors de Γ. Les équations (7) et (8) reproduisent, sur les séquences négatives, l'ordre total extrait des séquences positives. Cet ordre est reproduit en choisissant systématiquement les valeurs minimales envisageables. Si l'ordre total n'a pas pu être reproduit en sélectionnant les valeurs minimales alors il ne peut pas être reproduit avec les autres valeurs. L'équation (9) permet d'exclure les séquences négatives si la localisation du dernier patron de l'ordre total appartient à Γ. [2]

Objectif. L'objectif (10) est de maximiser le nombre de séquences exclues puis de minimiser le nombre de patrons sélectionnés.

4.2 Un modèle PPC pour l'ordre partiel

La figure 6 présente le modèle PPC pour la résolution lexicographique du problème MMP$\langle\{\mathtt{d_B}\}, \{\mathtt{cover}_{\mathcal{P}^+}, \mathtt{freq}_{\leq \mathtt{k}}, \mathtt{card}_{\leq 1}\}, \{\mathtt{partial}_{<,\alpha}\}\rangle$.

Variables. Pour tout $j \in [q]$, $c[j]$ est une variable booléenne qui réifie l'appartenance du patron j à la solution. l est une matrice de variables entières telle que chaque cellule $l[i,j] \in L[i,j]$ correspond à la localisation choisie pour le patron j dans la séquence i. a est une matrice de variables booléennes telle que chaque cellule $a[j,j']$ réifie la présence d'un arc de j vers j'. Pour tout $i \in \mathcal{P}^-$, $r[i]$ est une variable booléenne qui réifie l'exclusion de i.

Contraintes. L'équation (1) force la sélection d'au moins un patron dans l'ordre partiel. L'équation (2) impose que si j précède j', c'est-à-dire si $a[j,j'] = 1$, alors pour chaque séquence positive la localisation du patron j doit précéder la localisation du patron j' et réciproquement. L'équation (3) vérifie si l'ordre partiel est vérifié pour les séquences négatives. $check\_partial\_order$ effectue l'opération de vérification et affecte la valeur 0 a $r[i]$ si l'ordre partiel n'a pu être reconstitué. Si l'ordre partiel est totalement déterminé et que la séquence n'a pas pu être exclue alors r prend la valeur 1.

Objectif. L'objectif (4) est de maximiser le nombre de séquences exclues, puis de minimiser le nombre de patrons sélectionnés.

check_partial_order(c,a,D,r) est le co-propagateur utilisé dans l'équation (3). Il vérifie si une séquence peut satisfaire un ordre partiel. Une séquence est représentée par une liste d'ensembles (paramètre D) dont les éléments correspondent aux localisations des patrons dans la séquence. c est une liste de variables booléennes qui indique si un patron appartient à l'ordre partiel. a est une matrice de variables de booléennes qui représente la relation d'ordre entre les patrons. Le patron j précède le patron j' si $a[j,j'] = 1$. r est une variable booléenne qui indique si l'ordre partiel définit par le couple (c, r) peut être reproduit à partir des domaines fournis par D. Le co-propagateur utilise une valeur spéciale, noté $\epsilon = max(D) + 1$ et une liste de variables $starts$ telle que $\forall j \in [|c|]\ starts[j] \in D[j] \cup \{\epsilon\}$. Les algorithmes 1 et 2 montrent la mise à jour des valeurs de $starts$ Le premier algorithme est appelé lorsque qu'un patron est sélectionné ou exclu tandis que le second est appelé lorsque un arc est sélectionné ou exclu. Le principe est le même que celui utilisé pour l'ordre total : l'ordre partiel va être reproduit en utilisant les plus petites localisations possibles de D (ligne 2 de l'algorithme 1, ligne 3 de l'algorithme 2). Si l'un des $starts$ prend la valeur ϵ alors la séquence ne peut pas reproduire l'ordre partiel (lignes 3-5 de l'algorithme 1, lignes 4-6 de l'algorithme 2). Si tout les c et a sont assignés sans qu'aucun $starts$ ne prenne la valeur ϵ alors la séquence satisfait l'ordre (lignes 8-10 de l'algorithme 1, lignes 9-11 de l'algorithme 2).

1 propagate_column(j)

```
1.  if c[j] = 1 then
2.      starts[j] ← min(D[j] ∪ {ε})
3.      if starts[j] = ε then
4.          r ← 0
5.          End of propagation
6.      for all j' ∈ [|c|] s.t. a[j,j'] = 1 do
7.          propagate_arc(j,j')
8.  if all_assigned(a) and all_assigned(c) then
9.      r ← 0
10.     End of propagation
```

2 propagate_arcs(j,j')

```
1.  if a[j,j'] = 1 and c[j]=1 and c[j'] = 1 then
2.      if starts[j'] ≥ starts[j] then
3.          starts[j'] ← min({l|(l ∈ D[j] ∪ {ε}) ∧ (l >starts[j])})
4.          if starts[j'] = ε then
5.              r ← 0
6.              End of propagation
7.          for all j'' ∈ [|c|] s.t. a[j',j''] = 1 do
8.              propagate_arc(j',j'')
9.  if all_assigned(a) and all_assigned(c) then
10.     r ← 0
11.     End of propagation
```

1. Dans la contrainte globale $sort\_perm(src, perm, dest)$, le vecteur $dest$ est la version triée du vecteur src et $perm$ est la permutation qui permet de passer de src à $dest$.

2. La contrainte globale $min\_at\_least(var, Set, min)$ signifie que var prend la plus petite valeur de Set strictement supérieure à min.

4.3 Approche génétique

Nous présentons ici un algorithme génétique pour MMP$\langle\{\mathtt{d_B}\},\{\mathtt{cover}_{\mathcal{P}^+},\mathtt{freq}_{\leq \mathtt{k}},\mathtt{card}_{\leq 1}\},\{\mathtt{partial}_{<,\alpha}\}\rangle$ où l'on cherche à exclure le maximum de séquences avec un nombre de patrons minimal. Cet algorithme (Algorithme 3) se base sur le modèle PPC de la section 4.2 qui est utilisé pour générer la population initiale mais également croiser et fusionner les solutions. Les paramètres de l'algorithme sont $SizeP$ - le nombre d'individus de la population -, $SizeT$ - le nombre d'individus sélectionnés pour le tournoi (le nombre de fils générés est $SizeT/2$) -, It - le nombre d'itérations effectuées -, et To - une limite de temps utilisée par le modèle PPC. Le paramétrage utilisé pour les expérimentations est abordé en section 5.2.

3 Genetic$(\mathcal{P}^+, \mathcal{P}^-, SizeP, SizeT, It, To)$

1. $pop \leftarrow \varnothing$
2. **for each** $i \in \{1..SizeP\}$ **do**
3. $\quad pop \leftarrow pop \cup solve(\mathcal{P}^+, \mathcal{P}^-, To)$
4. **for each** $i \in \{1..It\}$ **do**
5. $\quad tourn \leftarrow selectNPair(pop, SizeT/2)$
6. $\quad new \leftarrow \varnothing$
7. $\quad$ **for each** $(p1, p2) \in tourn$ **do**
8. $\quad\quad tmp \leftarrow merge(p1, p2)$
9. $\quad\quad new \leftarrow new \cup solve(\mathcal{P}^+, \mathcal{P}^-, tmp, To)$
10. $\quad devTourn \leftarrow \{p | (p, \_) \vee (\_, p) \in tourn\}$
11. $\quad selected \leftarrow selectBest(new \cup devTourn)$
12. $\quad pop \leftarrow (pop \backslash devTourn) \cup selected$
13. **return** $best(pop)$

L'algorithme crée une population d'individus générés aléatoirement (lignes 1-3). A chaque itération, il sélectionne n paires d'individus distincts (ligne 5) et crée un nouvel individu pour chaque paire d'individus (lignes 6-9). Sont conservés les n meilleurs individus parmi les individus du tournoi et ceux créés (lignes 10-12). Enfin, le meilleur individu de la population est retourné (ligne 13).

Les appels à `solve` aux lignes 3 et 9 sont des appels au modèle PPC. Tous les problèmes transmis à ce modèle sont résolus jusqu'à l'obtention de la solution optimale ou lorsque la limite de temps imposée au modèle est atteinte. Les deux appels à `solve` retournent un individu qui est représenté par un ensemble de patrons (le vecteur c du modèle PPC) et un ensemble d'arcs (la matrice a du modèle PPC) qui représentent à eux deux l'ordre partiel extrait. La stratégie de branchement adoptée pour obtenir des individus différents dans la population est la suivante : les patrons sont sélectionnés les uns après les autres avec une probabilité de $0,5$, puis les arcs sont sélectionnés en ordre aléatoire avec une probabilité de $0,5$. L'ordre de sélection aléatoire pour les arcs est essentiel car la relation d'ordre partiel est antisymétrique et sélectionner les arcs dans un ordre déterministe pourrait empêcher l'exploration de certaines solutions. L'appel à `merge` (ligne 8) crée un nouvel individu à partir de deux individus connus. Le nouvel individu est créé par l'intersection des deux individus parents. L'intersection est définie comme suit : si un patron est présent (resp., absent) dans les deux individus, alors le patron est présent (resp., absent) dans l'individu fils ; l'intersection pour les arcs suit une procédure identique. Les patrons ou arcs qui sont présents dans un parent mais pas dans l'autre ne sont pas contraints, et c'est sur ces patrons et arcs que le modèle PPC effectuera les branchements.

(a) LEAP_12_0 (b) SHSP_17_3

FIGURE 7 – Deux exemples d'ordres partiels extraits.

5 Expérimentations

Cette section présente les résultats obtenus sur deux bases de séquences protéiques : Late Embryogenesis Abundant Proteins (LEAP) et Small Heat Shock Proteins (SHSP). La bases LEAP [5] comporte 1371 protéines réparties en 13 classes par des biologistes et la base SHSP [6] contient 3765 protéines réparties en 25 classes. Les programmes testés ont été implémentés avec la version 4.4.0 de la bibliothèque Gecode et les expérimentations ont été effectuées sur un processeur 2-core Intel Core i5-3380M (2,90GHz) avec une limite de 8Go de mémoire vive. Ces expérimentations ont été réalisées sur des instances contenant des patrons extraits des deux bases avec un algorithme ad hoc [9]. Un patron est défini comme une séquence consécutive de caractères (acides aminés) sur l'alphabet enrichi d'un caractère joker. Un patron ne peut commencer ni se terminer par un joker. Par exemple, le patron `A.B` couvre la séquence `CAAB` mais ne couvre pas la séquence `CADDB` car un joker se substitue à un seul caractère. Une instance comporte un ensemble de séquences scindé en séquences positives $\mathcal{P}^+$ et séquences négatives $\mathcal{P}^-$, un ensemble de patrons et les localisations de chaque patron dans chaque séquence. Tout patron possède au moins une localisation dans chaque séquence positive. Chacune des classes a été traitée tout à tour comme étant la classe positive, les séquences des autres classes formant alors la classe négative. L'objectif des expérimentations est d'extraire un ordre partiel, constitué des patrons définis dans les instances, qui couvre intégralement les séquences positives et qui exclue un maximum de séquences négatives. On utilise le terme *motif* en référence à un ordre partiel sur des patrons.

5.1 Approche complète

Le tableau 2 présente les résultats du modèle PPC sur 7 des classes de LEAP et 17 classes de SHSP. La première colonne correspond au nom de l'instance (le premier nombre correspond à la classe, tandis que le second correspond au nombre maximum de jokers consécutifs que

TABLE 2 – Résultats obtenus sur les bases LEAP et SHSP.

	Inst.	$\lvert\mathcal{P}^+\rvert$	$\lvert\mathcal{P}^-\rvert$	Pat.	$\lvert\mathcal{P}^-_{\varnothing}\rvert$	$\lvert c\rvert$	$\lvert a\rvert$	T (s)
LEAP	1_10*	208	1163	20	94	11	26	600
	2_8*	92	1279	25	16	12	25	600
	3_0	34	1337	9	0	1	0	0
	5_2	66	1305	10	0	3	3	3
	10_8*	78	1293	21	54	18	30	600
	11_1	35	1336	19	0	5	6	125
	12_0	17	1354	5	0	2	1	0
SHSP	2_6	109	3656	14	0	8	10	221
	3_2	64	3701	13	0	3	3	3
	4_0	23	3742	10	0	3	3	1
	5_0	25	3740	12	0	2	1	2
	7_10	100	3665	11	105	10	13	62
	8_1	30	3735	15	0	5	5	33
	13_3	156	3609	11	0	5	5	3
	14_0	84	3681	20	0	1	0	4
	15_0	26	3739	20	0	1	0	5
	16_9	75	3690	6	0	4	3	0
	17_3	88	3677	9	0	5	6	2
	19_1	23	3742	24	0	4	6	55
	21_2	131	3634	6	0	3	3	0
	22_0	15	3750	36	0	2	1	38
	23_4	60	3705	6	1	5	6	1
	24_2	107	3658	7	0	3	3	1
	25_2	57	3708	16	0	5	7	15

peuvent contenir les patrons). Les colonnes suivantes correspondent dans l'ordre : au nombre de séquences positives, au nombre de séquences négatives, au nombre de patrons, au nombre de séquences négatives qui n'ont pas été exclues par le motif trouvé (0 signifie que toutes les séquences négatives ont été exclues par le motif), au nombre de patrons qui composent le motif, au nombre d'arcs que contient le motif et au temps d'exécution en secondes. Les instances étiquetées avec une étoile sont celles dont le modèle n'a pas pu extraire le motif optimal en moins de dix minutes. La figure 7 présente deux exemples de motifs extraits.

Les motifs extraits pour 4 des 7 classes de la base LEAP et 15 des 17 classes de SHSP permettent une exclusion totale de la classe négative. De plus, dans certains cas l'ordre partiel permet d'exclure des séquences qui n'auraient pas pu être exclues avec un ordre total ou par un ensemble non ordonné de patrons.

5.2 Approche génétique

Nous présentons ici les comparaisons, sur les trois instances les plus difficiles, de trois méthodes : l'approche complète (section 4), une approche aléatoire et l'approche génétique (section 4.3). Les trois approches se basent sur le même modèle PPC et les mêmes heuristiques de branchement que celles utilisées dans l'approche génétique. L'approche aléatoire s'arrête lorsqu'une solution a été trouvé tandis que les deux autres approches s'arrêtent après un délai de 15 minutes.

L'approche complète et l'approche génétique ont chacune été exécutées 100 fois sur chaque instance et l'approche aléatoire 1000 fois. L'approche génétique a été exécutée sur une population de cent individus sur 150 itérations avec 10 solutions sélectionnées pour le tournoi lors de chaque itération. Les solutions qui constituent la population ont été initialisées avec l'approche complète mais avec une limite de temps de une seconde par solution. L'opérateur utilisé pour améliorer les solutions obtenus par fusion est également le modèle PPC mais avec une limite de temps de une demi-seconde, sauf lorsque le nombre de variables de décisions libres était inférieur à 15. Dans ce cas précis, le modèle a été exécuté jusqu'à obtenir la solution optimale.

TABLE 3 – Exclusion moyenne.

		Complete			Random			Evolutionary		
	Inst.	Avg	Min	Stdev	Avg	Min	Stdev	Avg	Min	Stdev
LEAP	1_10	69.0	41	14.0	85.3	51	22.8	38.3	36	4.5
	2_8	55.1	5	39.9	53.9	7	41.6	5.0	5	0.1
	10_8	50.4	35	10.9	75.7	34	29.8	29.9	28	1.1

Pour les trois instances, l'approche génétique est celle qui a obtenu les meilleurs résultats en terme de qualité de solution et de stabilité. L'écart type sur les deux autres méthodes sont importants. Une fois que les patrons sont choisis le modèle PPC a tendance à explorer le même voisinage car ces patrons ne sont jamais remis en cause, par exemple en redémarrant la recherche à partir de zéro. L'approche génétique évite cet inconvénient car la population est initialisée aléatoirement et permet ainsi d'explorer plus de solutions de l'espace de recherche.

6 État de l'art

Plusieurs auteurs ont proposé des modèles génériques, basés sur la PPC ou SAT, pour formuler des problèmes connus de fouille de données. Ces approches permettent de formuler de façon déclarative et efficace de tels problèmes. Deux problèmes principaux ont été étudiés : la recherche d'itemsets fréquents et la recherche de sous-séquences communes.

Recherche d'Itemsets. Dans [3], les auteurs proposent une approche basée sur la PPC pour extraire des itemsets fréquents d'une base de transactions. Ce problème peut être représenté par une matrice de 0 et de 1, dont les lignes représentent les transactions et les colonnes les items associés aux transactions. Un 1 dans une cellule de la matrice indique que l'item associé à la colonne de la cellule appartient à la transaction associée à la ligne de la cellule. L'objectif est alors d'extraire des matrices contenant uniquement des 1. Un itemset est dit fermé si la matrice extraite ne peut pas être étendue en ligne ou en colonne sans introduire de 0. Dans [4], les auteurs formulent plusieurs problèmes liés à la recherche d'itemsets fréquents (k-term DNF, k-clustering,...).

Recherche de sous-séquences. Il existe principalement deux formes de séquences extraites par les algorithmes de fouille de données. Pour les distinguer, on parlera de sous-séquences et de patrons. Une sous-séquence est une séquence de caractères où seul l'ordre d'apparition des carac-

tères importe, la distance entre deux caractères de la sous-séquence n'étant pas discriminante dans la recherche. Un patron est une séquence de caractères consécutifs où la distance entre les caractères joue un rôle discriminant. Souvent les patrons incluent un caractère joker qui représente tous les caractères de l'alphabet sur lequel les séquences sont encodées. Il existe principalement deux catégories de méthodes : celles qui exposent les enracinements des caractères [8, 12] et les autres [7, 11]. La première catégorie est plus adaptée à la recherche de patrons, mais elle autorise également la recherche de sous-séquences. Dans [11] les auteurs proposent un modèle de programmation par contraintes pour extraire des sous-séquences ou des patrons. L'approche utilisée se base sur la contrainte *regular* [15] et un encodage des séquences d'entrées sous la forme d'automates finis déterministes. Dans [12], les auteurs proposent deux contraintes globales pour effectuer la tâche d'extraction de sous-séquences ou patrons. La première contrainte vérifie uniquement la présence d'enracinements, tandis que la seconde expose les enracinements ce qui autorise d'imposer des contraintes sur la distance entre les caractères et donc d'extraire des patrons en plus des sous-séquences. Dans [7], les auteurs proposent une méthode pour extraire des sous-séquences. Comme pour les approches ad hocs présentées dans [2, 13, 14], les auteurs introduisent une contrainte globale qui permet d'énumérer efficacement les sous-séquences fréquentes avec une recherche en profondeur d'abord (propre au solver PPC).

7 Conclusion

Nous avons présenté le problème de décision MMP pour traiter différentes tâches de fouilles de données. Ce problème consiste à déterminer une matrice cohérente avec un ensemble de contraintes matricielles et maximale en ligne à cet égard. Nous avons présenté deux variantes du problème qui ont des applications en bioinformatique : la recherche de patrons partiellement ou totalement ordonnés par leur localisations et permettant de discriminer des classes de séquences prédéfinies. Ces problèmes sont NP-complet et nous avons présenté deux modèles PPC ainsi qu'un algorithme génétique. Les résultats expérimentaux sur données biologiques sont prometteurs. Nous nous proposons par la suite de développer des co-propagateurs pour un spectre plus large de contraintes matricielles et également d'étudier différents algorithmes et stratégies de recherche afin d'améliorer le passage à l'échelle.

Références

[1] Emmanuel Coquery, Said Jabbour, Lakhdar Saïs, and Yakoub Salhi. A SAT-Based approach for discovering frequent, closed and maximal patterns in a sequence. In *ECAI*, pages 258–263, 2012.

[2] Christie I Ezeife and Yi Lu. Mining web log sequential patterns with position coded pre-order linked wap-tree. *DMKD*, 10(1) :5–38, 2005.

[3] Tias Guns, Siegfried Nijssen, and Luc De Raedt. Itemset mining : A constraint programming perspective. *AI*, 175(12–13) :1951–1983, 2011.

[4] Tias Guns, Siegfried Nijssen, and Luc De Raedt. k-pattern set mining under constraints. *Trans. KDE*, 25(2) :402–418, 2013.

[5] Gilles Hunault and Emmanuel Jaspard. The late embryogenesis abundant proteins DataBase. http://forge.info.univ-angers.fr/~gh/Leadb/index.php, 2013.

[6] Gilles Hunault and Emmanuel Jaspard. The small heat shock proteins database. sHSPdb. http://forge.info.univ-angers.fr/~gh/Shspdb/index.php, 2013.

[7] Amina Kemmar, Samir Loudni, Yahia Lebbah, Patrice Boizumault, and Thierry Charnois. Prefix-projection global constraint for sequential pattern mining. *arXiv preprint arXiv :1504.07877*, 2015.

[8] Amina Kemmar, Willy Ugarte, Samir Loudni, Thierry Charnois, Yahia Lebbah, Patrice Boizumault, and Bruno Cremilleux. Mining relevant sequence patterns with cp-based framework. In *ICTAI*, pages 552–559. IEEE, 2014.

[9] David Lesaint, Deepak Mehta, Barry O'Sullivan, and Vincent Vigneron. A Decomposition Approach for Discovering Discriminative Motifs in a Sequence Database. In *ECAI*, Prague, Czech Republic, 2014.

[10] J.-P. Métivier, S. Loudni, and T. Charnois. A Constraint Programming Approach for Mining Sequential Patterns in a Sequence Database. In *LML'13*, pages 1–15, 2013.

[11] Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni. A constraint language for declarative pattern discovery. In *ACM*, SAC '12, page 119–125, New York, NY, USA, 2012. ACM.

[12] Benjamin Negrevergne and Tias Guns. Constraint-based sequence mining using constraint programming. *arXiv preprint arXiv :1501.01178*, 2015.

[13] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan : Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE*, pages 0215–0215. IEEE, 2001.

[14] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, and Hua Zhu. Mining access patterns efficiently from web logs. In *KDDM. Current Issues and New Applications*, pages 396–407. Springer, 2000.

[15] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In *CP*, pages 482–495. Springer, 2004.

Réduction et Encodage des Contraintes Ensemblistes en SAT

Frédéric Lardeux[1] **Eric Monfroy**[2]

[1] LERIA - Université d'Angers, Angers, France.
[2] LINA, UMR CNRS 6241, TASC INRIA, Université de Nantes, Nantes, France.
Frederic.Lardeux@univ-angers.fr Eric.Monfroy@univ-nantes.fr

Résumé

D'un côté, les problèmes de satisfaction de contraintes (CSP) procurent une méthode déclarative et expressive pour modéliser les problèmes. D'un autre côté, les solveurs pour les problèmes de satisfiabilité de formules logique propositionnelle (SAT) peuvent manipuler des instances énormes jusqu'à des millions de clauses et variables. Dans cet article, nous présentons une approche bénéficiant de la modélisation CSP et de la résolution SAT. Notre technique consiste à modéliser, de façon expressive, des problèmes de contraintes ensemblistes en en CSP qui sont ensuite automatiquement réduits afin de retirer les valeurs des variables qui ne participent à aucune solution. Ces CSPs réduits sont ensuite encodés en de "bonnes" instances SAT qui peuvent être résolues par des solveurs SAT standards. Nous illustrons notre technique par divers problèmes standards : le Sudoku, le Social Golfer Problem et le Sports Tournament Scheduling Problem.

Notre technique est plus simple, plus expressive et moins sensible aux erreurs qu'une modélisation directe en SAT. De plus, les instances SAT automatiquement générées sont généralement plus petites que celles directement écrite pour un problème particulier (comme par exemple pour le Social Golfer Problem [18]) et peuvent être évaluées efficacement même pour des instances énormes. Enfin, la phase de réduction nous permet de repousser les limites et de traiter des problèmes encore plus gros.

Abstract

On the one hand, Constraint Satisfaction Problems (CSP) are a declarative and expressive approach for modeling problems. On the other hand, propositional satisfiability problem (SAT) solvers can handle huge SAT instances up to millions of variables and clauses. In this article, we present an approach for taking advantage of both CSP modeling and SAT solving. Our technique consists in expressively modeling set constraint problems as CSPs that are automatically treated by some reduction rules to remove values that do not participate in any solution. These reduced CSPs are then encoded into "good" SAT instances that can be solved by standard SAT solvers. We illustrate our technique on various well-known problems such as Sudoku, the Social Golfer problem, and the Sports Tournament Scheduling problem. Our technique is simpler, more expressive, and less error-prone than direct SAT modeling. The SAT instances that we automatically generate are rather small (even w.r.t. direct-written SAT instances for the Social Golfer problem [18]) and can efficiently be solved up to huge instances. Moreover, the reduction phase enables to push back the limits and treat even bigger problems.

1 Introduction

Une méthode classique consiste à formuler les problèmes combinatoires comme des problèmes de satisfaction de contraintes (CSP) [21]. Un CSP est défini par des variables et des contraintes reliant ces variables. Résoudre un CSP consiste à trouver une affectation des variables qui satisfait toutes les contraintes. L'expressivité est l'une des forces des CSP : les variables et les contraintes peuvent être de types variés. De plus, les contraintes globales améliorent non seulement la résolution mais aussi l'expressivité : elles apportent de nouveaux constructeurs et relations tels que Alldifferent (pour forcer toutes les variables d'une liste à avoir des valeurs différentes), Cardinalité (pour relier un ensemble à sa taille), ...

Une autre façon de formuler les problèmes combinatoires est d'utiliser le problème satisfiabilité d'une formule logique propositionnelle (SAT) [12]. Un problème SAT est une formule booléenne sous forme normale conjonctive, c'est à dire une conjonction de clauses.

Une clause est une disjunction de litéraux et un litéral est une variable ou sa négation. Quand toutes les clauses peuvent être satisfaites, le problème est dit satisfiable. En terme d'expressivité SAT est donc réduit à des variables booléennes et des formules propositionnelles. Coder des contraintes directement en SAT est une tâche pénible (par exemple [22] ou[13]). De plus, lorsque l'on essaie d'optimiser son modèle (en terme de variables et clauses), cela conduit rapidement à des modèles très complexes et illisibles dans lesquels apparaissent facilement des erreurs. Cependant, les solveurs SAT peuvent maintenant manipuler des instances énormes (des millions de variables). Il est donc intéressant de 1) encoder les CSP en SAT (e.g., [4, 6]) afin de bénéficier de l'expressivité des CSP et de la puissance de SAT, et 2) d'introduire plus d'expressivité en SAT, e.g., avec des contraintes globales telles que Alldifferent[17] ou Cardinalité[5].

Divers systèmes de contraintes ensemblistes ([19],[14],[1]) ont été réalisés et il a été montré que de nombreux problèmes peuvent facilement être modélisés par des contraintes ensemblistes.

Nous nous intéressons ici à l'encodage de contraintes ensemblistes en instances SAT. Dans [18], nous avons présenté des règles d'encodage qui sont appliquées directement sur les contraintes ensemblistes CSP. Cependant, nous avons remarqué que ces contraintes ensemblistes ne sont pas toujours aussi petites qu'elles pourraient l'être : des éléments possiblement dans les ensembles peuvent être retirés sans perte de solutions. Ainsi, les instances SAT générées peuvent aussi être réduites.

Il est inconcevable d'assurer que le programmeur écrira des modèles CSP "minimaux", donc notre approche consiste à procurer :

- un ensemble de contraintes simples mais complètes et expressives (intersection, union, cardinal d'ensembles, ...)
- un ensemble de règles de réduction ($\Rightarrow_{red}$) réduisant les modèles CSP par propagation de contraintes [2]. La consistance calculée porte sur les bornes et cardinalité des ensembles (similaire à celle présentée dans [3]) qui est plus forte que la consistance de borne [20].
- un ensemble de règles d'encodage ($\Leftrightarrow_{enc}$) convertissant des contraintes CSP en instances SAT.

Dans ce papier, nous illustrons notre approche avec le Social Golfer Problem[18], le Sports Tournament Scheduling problem [23] ainsi que le Sudoku. Nous avons aussi appliqué notre approche à d'autres problèmes (tels que le Car-sequencing et le WhoWithWhom, ...) et les instances SAT générées automatiquement ne sont pas plus complexes que celles générées et optimisées directement et leur résolution avec un solveur SAT classique (dans notre cas Minisat [10]) est efficace.

Comparé à [18], les règles de réduction $\Rightarrow_{red}$ nous permettent :

- d'obtenir un encodage simplifié ;
- d'obtenir des instances plus petites ; ces problèmes sont résolus plus rapidement ;
- d'aborder et de résoudre des problèmes plus grands que nous étions incapable d'encoder auparavant pour des raisons de taille.

Les travaux sur les techniques d'encodage telles que [4] et [6] établissent une relation entre la résolution CSP et SAT en terme de propriétés telles que les consistances. Nous nous intéressons ici à un autre type de contraintes (i.e., contraintes ensemblistes) et nous tentons d'obtenir de petites instances SAT aux propriétés adaptées aux solveurs SAT. De plus, [4] et [6] ne considèrent pas de règles de réduction comme nos règles $\Rightarrow_{red}$. Notre approche est similaire à [17] où les contraintes globales Alldifferent et des superpositions de contraintes Alldifferent sont manipulées de manières expressives avant d'être encodées automatiquement en SAT par des techniques de réécritures.

Il est à noter que nous utilisons les travaux présentés dans [5] à propos de la contrainte globale Cardinalité afin d'améliorer l'encodage de la cardinalité d'un ensemble. Notre phase de réduction est plus forte que celle du solveur Conjunto [14] et similaire à celle de [8]. Dans [14], la propagation est réalisée à partir des bornes inférieures et supérieures des ensembles. Dans nos règles $\Rightarrow_{red}$ ainsi que dans [8], la propagation est basée sur les bornes inférieure et supérieure mais aussi sur les cardinalités minimum et maximum des ensembles. Il faut aussi noter que dans notre cas, la réduction est utilisée comme pré-traitement avant l'encodage en instances SAT alors que dans Conjunto et dans [8], la réduction est utilisée conjointement à l'énumération afin de produire un solveur complet. Notre objectif n'est pas de résoudre le modèle mais de le traiter afin d'obtenir un meilleur encodage que sera résolu par un solveur SAT. Nous ne cherchons pas à obtenir de meilleurs résultats que les solveurs CSP ensemblistes mais à introduire les contraintes ensemblistes dans SAT.

2 Problème de Satisfaction de Contraintes avec Ensembles

2.1 CSP Ensembliste

Definition 1 (CSP-Ensembliste)
Un CSP-Ensembliste est défini par :

- *un univers $\mathcal{U}$ d'entiers.*

— *un ensemble X de variables entières telles qu'un domaine $D_x \subseteq \mathcal{U}$ est associé à chaque variable entière $x \in \mathcal{X}$.*
— *un ensemble $\mathbb{F}$ de variables ensemblistes telles que pour chaque variable ensembliste $F \in \mathbb{F}$ sont associées :*
  — *une borne inférieure $\mathfrak{f} \subseteq \mathcal{U}$*
  — *une borne supérieure $\mathfrak{F} \subseteq \mathcal{U}$*
  — *une cardinalité minimum $\#_{\downarrow}F \in \mathbb{N}$*
  — *une cardinalité maximum $\#_{\uparrow}F \in \mathbb{N}$*
  Nous avons comme propriétés que : $\mathfrak{f} \subseteq F \subseteq \mathfrak{F}$, $\#_{\downarrow}F \leq \#F \leq \#_{\uparrow}F$, $\#\mathfrak{f} \leq \#_{\downarrow}F$, et $\#\mathfrak{F} \geq \#_{\uparrow}F$ où $\#F$ représente la cardinalité de F.
— *un ensemble de contraintes C qui sont des relations définies sur $\mathcal{D}^{|X|} \times \mathcal{U}^{|\mathbb{F}|}$.*

Il est à noter que la borne inférieure d'une variable ensembliste $F \in \mathbb{F}$ représente des valeurs (entiers) qui sont effectivement dans F ; la borne supérieure représente des valeurs qui sont possiblement (ou sont effectivement si elle sont également dans la borne inférieure) dans F. Nous avons donc $\mathfrak{f} \subseteq F \subseteq \mathfrak{F}$ et les valeurs possibles de F sont les éléments de l'ensemble des parties $2^{\mathfrak{F}}$ contenant $\mathfrak{f}$ et tels que leur cardinalité soit comprise dans $[\#_{\downarrow}F..\#_{\uparrow}F]$.

2.2 Contraintes Ensemblistes de Base

Les variables sont déclarées et initialisées de la manière suivante :
— un intervalle I d'entiers est noté $n..m$ où n et m sont des entiers ; $I = n..m$ représente tous les entiers compris entre n et m ; la borne inférieure de l'intervalle I est notée $\underline{I}$ et la borne supérieure $\overline{I}$.
— un s_intervalle est une séquence ordonnée d'intervalles disjoints [1] :

$$Li = (L_1, \ldots, L_l)$$

Le s_intervalle vide est noté $\perp$. Un s_intervalle $sI = (I_1, \ldots, I_n)$ est inclu dans un s_intervalle $sJ = (J_1, \ldots, J_m)$ si chaque entier apparaissant dans sI apparait aussi dans sJ :

$$sI \subseteq sJ \;\Leftrightarrow\; \forall I_i, \forall v \in [\underline{I_i}..\overline{I_i}], \exists J_k, \underline{J_k} \leq v \leq \overline{J_k}$$

La cardinalité $\#sI$ d'un s_intervalle $sI = (I_1, \ldots, I_n)$ est donnée par : $\#sI = \sum_{i=1}^{n}(\overline{I_i} - \underline{I_i} + 1)$. Les autres opérations (comme $\cup$, $\cap$, ...) sur les s_intervalles peuvent être obtenues de manière similaire.
— l'univers $\mathcal{U}$ est déclaré comme $\boldsymbol{\mathcal{U}} :: \top$ où $\top$ est un s_intervalle ;
— une variable entière x est déclarée comme $\boldsymbol{x :: D_x}$ où son domaine D_x est un s_intervalle ;
— une variable ensembliste F est déclarée comme $\boldsymbol{F :: \mathfrak{f}, \mathfrak{F}, \#_{\downarrow}F, \#_{\uparrow}F}$ où la borne inférieure $\mathfrak{f}$ et sa borne supérieure $\mathfrak{F}$ sont des s_intervalles, et la cardinalité minimum $\#_{\downarrow}F$ et la cardinalité maximum $\#_{\uparrow}F$ sont des entiers.

Une variable ensembliste particulière est celle représentant l'ensemble vide. Elle est notée $\emptyset$ et est définie par $\emptyset :: \perp, \perp, 0, 0$

Considérons F, G, H et F_i (i compris entre 1 et n) étant des variables ensemblistes et x étant une variable entière. Nous énumérons ici plusieurs contraintes ensemblistes courantes que nous avons traitées :

element (dis)equality	$x = y$	$(x \neq y)$
(non)membership	$x \in F$	$(x \notin F)$
set (dis)equality	$F = G$	$(F \neq G)$
intersection	$H = F \cap G$	
union	$H = F \cup G$	
disjoint union	$H = F \sqcup G$	
inclusion	$F \subseteq G$	$(F \not\subseteq G)$
difference	$H = F \setminus G$	
multi-intersection	$F = \bigcap_{i=1}^{n} F_i$	
multi-union	$F = \bigcup_{i=1}^{n} F_i$	
partition	$F = \bigsqcup_{i=1}^{n} F_i$	
cardinality	$x = \lvert F \rvert$	
minimum $<$	$x = min(F)$	
maximum $>$	$x = max(F)$	

3 Règles de Réduction

Les règles de réduction $\Rightarrow_{red}$ ont pour but de réduire l'espace de recherche. Elles peuvent donc ajouter des valeurs aux bornes inférieures, supprimer des valeurs aux bornes supérieures, augmenter la cardinalité minimale ou diminuer la cardinalité maximale des variables ensemblistes traitées. La consistance calculée par nos règles est similaire à la consistance calculée par le filtrage de [3] pour les ensembles.

Pour les variables entières, nos règles ne peuvent que supprimer des valeurs de leur domaine. Enfin, certaines règles peuvent aussi amener à un échec.

3.1 Variables Entières

Si une variable entière x a un domaine vide, le CSP n'a pas de solution :

$$D_x = \perp \;\Rightarrow_{red}\; echec \tag{1}$$

Quand une variable entière x est déclarée deux fois, les deux déclarations sont groupées en une seule :

$$x :: D_x, x :: D' \;\equiv\; x :: D_x \cap D'_x \tag{2}$$

1. Soit $I = a..b$ et $J = c..d$. $I \prec J$ si $b < c$.

Notons que l'application de la règle 2 remplace $x :: D_x$ et $x :: D'_x$ par $x :: D_x \cap D'_x$.

3.2 Variables Ensemblistes

$$\#_{\downarrow}F > \#_{\uparrow}F \;\Rightarrow_{red}\; echec \tag{3}$$
$$\mathfrak{f} \not\subseteq \mathfrak{F} \;\Rightarrow_{red}\; echec \tag{4}$$
$$\#\mathfrak{f} > \#_{\uparrow}F \;\Rightarrow_{red}\; echec \tag{5}$$
$$\#\mathfrak{F} < \#_{\downarrow}F \;\Rightarrow_{red}\; echec \tag{6}$$

La règle 4 est très importante car elle garantit que toute valeur présente dans $\mathfrak{f}$ est dans $\mathfrak{F}$ et que dans le cas contraire un cas d'échec est constaté. Les règles 4, 5 et 6 sont très utiles quand une contrainte de cardinalité modifie les cardinalités minimum et maximum d'une variable ensembliste.

La règle 7 signifie que F est l'ensemble vide (si $\mathfrak{f} \neq \bot$ alors cela conduira à un cas d'échec avec la règle 4) :

$$\#_{\uparrow}F = 0, \;\; \mathfrak{F} \neq \bot \;\Rightarrow_{red}\; \mathfrak{F} = \bot \tag{7}$$

Les règles 8 et 9 rendent la variable ensembliste F constante quand la taille de sa borne supérieure est égale à sa cardinalité minimum ou quand la taille de sa borne inférieure est égale à sa cardinalité maximum :

$$\#_{\downarrow}F = \#\mathfrak{F}, \;\; \mathfrak{f} \subset \mathfrak{F}, \;\; \#_{\downarrow}F \leq \#_{\uparrow}F \;\Rightarrow_{red}\; \left\{ \begin{array}{l} \mathfrak{f} \leftarrow \mathfrak{F} \\ \#_{\uparrow}F \leftarrow \#_{\downarrow}F \end{array} \right. \tag{8}$$

$$\#_{\uparrow}F = \#\mathfrak{f}, \;\; \mathfrak{f} \subset \mathfrak{F}, \;\; \#_{\downarrow}F \leq \#_{\uparrow}F \;\Rightarrow_{red}\; \left\{ \begin{array}{l} \mathfrak{F} \leftarrow \mathfrak{f} \\ \#_{\downarrow}F \leftarrow \#_{\uparrow}F \end{array} \right. \tag{9}$$

Les règles suivantes ne peuvent être déclenchées que s'il y a une erreur dans la déclaration de la variable ensembliste :

$$\#\mathfrak{f} > \#_{\downarrow}F \;\Rightarrow_{red}\; \#_{\downarrow}F \leftarrow \#\mathfrak{f} \tag{10}$$
$$\#\mathfrak{F} < \#_{\uparrow}F \;\Rightarrow_{red}\; \#_{\uparrow}F \leftarrow \#\mathfrak{F} \tag{11}$$

3.3 Contraintes Ensemblistes

Nous n'avons pas assez d'espace pour présenter les réductions ainsi que les règles de transformation pour toutes les contraintes ensemblistes. Nous nous contentons donc de ne présenter que les règles de réduction pour l'union de deux d'ensembles.

$H = F \cap G$
$\Rightarrow_{red}$

$$\left\{ \begin{array}{lcl} \mathfrak{H}' & \leftarrow & \mathfrak{H} \cap \mathfrak{F} \cap \mathfrak{G} \\ \mathfrak{h}' & \leftarrow & \mathfrak{h} \cup (\mathfrak{f} \cap \mathfrak{g}) \\ \mathfrak{f}' & \leftarrow & \mathfrak{f} \cup \mathfrak{h} \\ \mathfrak{g}' & \leftarrow & \mathfrak{g} \cup \mathfrak{h} \\ \mathfrak{F}' & \leftarrow & \mathfrak{F} \setminus (\mathfrak{g} \setminus \mathfrak{H}) \\ \mathfrak{G}' & \leftarrow & \mathfrak{G} \setminus (\mathfrak{f} \setminus \mathfrak{H}) \\ \#_{\downarrow}F' & \leftarrow & max\{\#_{\downarrow}F, \#\mathfrak{f}', \#_{\downarrow}H\} \\ \#_{\downarrow}G' & \leftarrow & max\{\#_{\downarrow}G, \#\mathfrak{g}', \#_{\downarrow}H\} \\ \#_{\downarrow}H' & \leftarrow & max\{\#_{\downarrow}H, \; \#\mathfrak{h}'\} \\ \#_{\uparrow}F' & \leftarrow & min\{\#_{\uparrow}F, \#\mathfrak{F}'\} \\ \#_{\uparrow}G' & \leftarrow & min\{\#_{\uparrow}G, \#\mathfrak{G}'\} \\ \#_{\uparrow}H' & \leftarrow & min\{\#_{\uparrow}H, \#_{\uparrow}F, \#_{\uparrow}G, \#\mathfrak{H}'\} \end{array} \right. \tag{12}$$

Quand les 3 ensembles sont fixés, i.e., $\mathfrak{h} = \mathfrak{f} \cap \mathfrak{g} = \mathfrak{H} = \mathfrak{F} \cap \mathfrak{G}$ alors la contrainte $H = F \cap G$ peut être supprimée.

La règle suivante est appliquée quand H est fixé. Pour simplifier la notation, nous notons $\bar{\mathfrak{f}}$ la variable ensembliste correspondant à $\mathfrak{F} \setminus \mathfrak{f}$.

$$\begin{array}{l} H = F \cap G, \\ \mathfrak{h} = \mathfrak{H}, \\ \#_{\downarrow}H = \#_{\uparrow}H \end{array} \Rightarrow_{red} \left\{ \begin{array}{lcl} \mathfrak{H}' & \leftarrow & \mathfrak{H} \\ \mathfrak{h}' & \leftarrow & \mathfrak{h} \\ \mathfrak{f}' & \leftarrow & \mathfrak{f} \cup \mathfrak{h} \\ \mathfrak{g}' & \leftarrow & \mathfrak{g} \cup \mathfrak{h} \\ \mathfrak{F}' & \leftarrow & \mathfrak{F} \setminus (\bar{\mathfrak{f}}' \cap \mathfrak{g}') \\ \mathfrak{G}' & \leftarrow & \mathfrak{G} \setminus (\bar{\mathfrak{g}}' \cap \mathfrak{f}') \\ \#_{\downarrow}F' & \leftarrow & max\{\#_{\downarrow}F, \#\mathfrak{f}'\} \\ \#_{\downarrow}G' & \leftarrow & max\{\#_{\downarrow}G, \#\mathfrak{g}'\} \\ \#_{\downarrow}H' & \leftarrow & \#_{\downarrow}H \\ \#_{\uparrow}F' & \leftarrow & min\{\#_{\uparrow}F, \#\mathfrak{F}'\} \\ \#_{\uparrow}G' & \leftarrow & min\{\#_{\uparrow}G, \#\mathfrak{G}'\} \\ \#_{\uparrow}H' & \leftarrow & \#_{\uparrow}H \end{array} \right. \tag{13}$$

Nous supposons maintenant que les règles suivantes ont été appliquées pour la contrainte d'intersection. Nous pouvons donc garantir que :

- $\#_{\downarrow}F - \#\mathfrak{f}$ est le nombre minimum de variables entières requis dans F ;
- $\#_{\uparrow}F - \#\mathfrak{f}$ est le nombre maximum de variables entières encore acceptable dans F ;
- $\#(\bar{\mathfrak{f}} \setminus \mathfrak{G})$ est le nombre de variables entières potentiellement dans F n'influençant pas H ;
- $\#(\bar{\mathfrak{f}} \setminus \mathfrak{g})$ est le nombre de variables entières potentiellement dans F influençant H ;
- $\#(\bar{\mathfrak{f}} \cap \bar{\mathfrak{g}})$ est le nombre de variables entières potentiellement dans F pouvant influencer H ;
- $\#_{\downarrow}F - \#\mathfrak{f} - \#(\bar{\mathfrak{f}} \setminus \mathfrak{G}) - \#(\bar{\mathfrak{f}} \cap \bar{\mathfrak{g}})$ est le nombre minimum de variables entières requis dans F et qui effectivement modifie H ;
- $\#_{\uparrow}H - \#\mathfrak{h}$ est le nombre maximum de variables entières encore acceptable dans H.

Nous avons ajouté une règle pour le cas spécial de la variable ensembliste vide. Cette règle n'est pas obligatoire mais elle est plus simple et plus lisible que la

précédente. De plus, elle est utile pour la contrainte d'union disjoint :

$$\begin{array}{l}\emptyset = F \cap G \\ \Rightarrow_{red}\end{array}$$

$$\left\{\begin{array}{lcl} \mathfrak{f}' & \leftarrow & \mathfrak{f} \\ \mathfrak{g}' & \leftarrow & \mathfrak{g} \\ \mathfrak{F}' & \leftarrow & \mathfrak{F} \setminus \mathfrak{g} \\ \mathfrak{G}' & \leftarrow & \mathfrak{G} \setminus \mathfrak{f} \\ \#_{\downarrow}F' & \leftarrow & \#_{\downarrow}F \\ \#_{\downarrow}G' & \leftarrow & \#_{\downarrow}G \\ \#_{\uparrow}F' & \leftarrow & \min\{\#_{\uparrow}F, \#(\mathfrak{F} \setminus \mathfrak{g})\} \\ \#_{\uparrow}G' & \leftarrow & \min\{\#_{\uparrow}G, \#(\mathfrak{G} \setminus \mathfrak{f})\} \end{array}\right. \quad (14)$$

4 Règles d'Encodage

Les règles d'encodage $\Leftrightarrow_{enc}$ ont pour but de transformer les contraintes ensemblistes d'un CSP en clauses SAT. Nos règles fonctionnent à la fois sur des instances réduites et non réduites de contraintes ensemblistes.

4.1 Variables Entières

Cette règle d'encodage force chaque variable entière à n'avoir qu'une et une seule valeur de son domaine.

$$\begin{array}{c} VariableEntiere(v, D_v) \\ \Leftrightarrow_{enc} \\ \forall x \in D_v,\ x_v \rightarrow \bigwedge_{y \in D_v, x \neq y} \neg y_v \ \ and \ \ \bigvee_{x \in D_v} x_v \\ \Leftrightarrow \\ (\bigwedge_{x \in D_v} \bigwedge_{y \in D_v, y > x} (\neg x_v \vee \neg y_v)) \wedge \bigvee_{x \in D_v} x_v \end{array}$$

Chaque encodage de variable entière génère $|D_v|.(|D_v|-1)/2$ cl. binaires et 1 $|D_v|aire$ clause.

4.2 Variables Ensemblistes

Pour une variable ensembliste $F :: (\mathfrak{f}, \mathfrak{F}, \#_{\downarrow}F, \#_{\uparrow}F)$, l'encodage consiste à créer des variables booléennes pour chaque valeur de la borne supérieure $\mathfrak{F}$ et de mettre à vrai celles appartenant à la borne inférieure $\mathfrak{f}$. Si nous considérons une constante x de l'univers $\mathcal{U}$, nous notons $?x_{\mathcal{F}}$ la création de la variable booléenne $x_{\mathcal{F}}$ représentant l'appartenance de x à la variable ensembliste F.

$$\begin{array}{c} F :: (\#_{\downarrow}F, \#_{\uparrow}F, \mathfrak{f}, \mathfrak{F}) \\ \Leftrightarrow_{enc} \\ \left\{\begin{array}{l} \forall x \in \mathfrak{F},\ ?x_{\mathcal{F}} \\ \forall x \in \mathfrak{f},\ x_{\mathcal{F}} \end{array}\right. \end{array}$$

4.3 Intersection d'Ensembles

Afin d'être complet, nous considérons tous les cas par rapport à $\mathfrak{H}, \mathfrak{h}, \mathfrak{F}, \mathfrak{f}, \mathfrak{G}, \mathfrak{g}$, bien que de nombreux cas soient impossibles. La table 1 détaille cette règle d'encodage avec en face de chaque cas le nombre et la forme des clauses générées.

5 Résultats expérimentaux

Toutes les expériences ont été réalisées sur un processeur Intel® Xeon® E5-2670 avec 2.3GHz et 230 Go RAM. Les règles d'encodage ont été implémentées en C^{++} et celles de réduction comme des Constraint Handling Rules (CHR[11]). Nous avons utilisé Minisat [10] comme solveur SAT pour toutes nos expérimentations.

5.1 Problèmes encodés

5.1.1 Sudoku

Le problème du Sudoku 9×9 [2] est un puzzle où chaque ligne, chaque colonne et chaque bloc doivent contenir toutes les valeurs entre 1 et 9. Des instances difficiles sont accessibles sur un site web [3] sur lequel elles sont référencées par un numéro de grille.

5.1.2 Social Golfer Problem

Le problème du Social Golfer (problème numéro 10 de la CSPLib[16]) est le suivant : q golfeurs jouent toutes les semaines durant w semaines en se séparant en g groupes de p golfeurs ($q = p.g$). Comment organiser la partie de ces golfeurs afin qu'aucun golfeur ne joue dans le même groupe qu'un autre golfeur plus d'une fois ? Une instance de ce problème est donc caractérisée par un triplet $g - p - w$. De nombreuses instances du Social Golfer sont encore ouvertes et ce problème est attractif puisqu'il est en relation avec des problèmes de cryptage et de couverture. Nous avons testé deux modélisations possibles pour la *contrainte de socialisation* du problème [4]. La première utilise des contraintes de cardinalité (le nom des instances est suffixée par "_C") et la seconde des contraintes d'implication (le nom des instances est suffixée par "_I"). Les differents modèles peuvent être trouvés dans [18].

5.1.3 Sports Tournament Scheduling

Ce problème a été proposé par Toby Walsh (problème numéro 26 de la CSPLib [23]) de la manière suivante : Il faut organiser un tournoi avec n équipes sur $n-1$ semaines où chaque semaine est divisée en $n/2$ périodes et où chaque période est divisée en deux créneaux. La première équipe de chaque créneau

2. http ://en.wikipedia.org/wiki/Sudoku
3. http ://www.e-sudoku.fr/jouer-sudoku-solo.php
4. Deux joueurs ne peuvent jouer plus d'une fois dans un même groupe

$$
\begin{array}{llll}
 & & & H = F \cap G \\
 & & & \Leftrightarrow_{enc} \\
\forall x \in \mathcal{U} & x \in \mathfrak{h} & x \in \mathfrak{f} & x \in \mathfrak{g} \quad true \\
 & & & x \in \mathfrak{G} \setminus \mathfrak{g} \quad x_{\mathcal{G}} \quad \#(\mathfrak{h} \cap \mathfrak{f} \cap \bar{\mathfrak{g}}) \text{ cl. unitaires} \\
 & & & x \notin \mathfrak{G} \quad false \\
 & & x \in \mathfrak{F} \setminus \mathfrak{f} & x \in \mathfrak{g} \quad x_{\mathcal{F}} \quad \#(\mathfrak{h} \cap \bar{\mathfrak{f}} \cap \mathfrak{g}) \text{ cl. unitaires} \\
 & & & x \in \mathfrak{G} \setminus \mathfrak{g} \quad x_{\mathcal{F}} \wedge x_{\mathcal{G}} \quad \#(\mathfrak{h} \cap (\mathfrak{F} \setminus \mathfrak{f}) \cap \bar{\mathfrak{g}}) \ 2\times \text{ cl. unitaires} \\
 & & & x \notin \mathfrak{G} \quad false \\
 & & x \notin \mathfrak{F} & x \in \mathfrak{g} \quad false \\
 & & & x \in \mathfrak{G} \setminus \mathfrak{g} \quad false \\
 & & & x \notin \mathfrak{G} \quad false \\
 & x \in \mathfrak{H} \setminus \mathfrak{h} & x \in \mathfrak{f} & x \in \mathfrak{g} \quad x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \cap \mathfrak{f} \cap \mathfrak{g}) \text{ cl. unitaires} \\
 & & & x \in \mathfrak{G} \setminus \mathfrak{g} \quad x_{\mathcal{H}} \leftrightarrow x_{\mathcal{G}} \quad \#(\bar{\mathfrak{h}} \cap \mathfrak{f} \cap \bar{\mathfrak{g}}) \times 2 \text{ cl. binaires} \\
 & & & x \notin \mathfrak{G} \quad \neg x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \cap (\mathfrak{f} \setminus \mathfrak{G})) \text{ cl. unitaires} \\
 & & x \in \mathfrak{F} \setminus \mathfrak{f} & x \in \mathfrak{g} \quad x_{\mathcal{H}} \leftrightarrow x_{\mathcal{F}} \quad \#(\bar{\mathfrak{h}} \cap \bar{\mathfrak{f}} \cap \mathfrak{g}) \times 2 \text{ cl. binaires} \\
 & & & x \in \mathfrak{G} \setminus \mathfrak{g} \quad x_{\mathcal{H}} \leftrightarrow x_{\mathcal{F}} \wedge x_{\mathcal{G}} \quad \#(\bar{\mathfrak{h}} \cap \bar{\mathfrak{f}} \cap \bar{\mathfrak{g}}) \times 2 \text{ cl. binaires } et \text{ 1 cl. ternaire} \\
 & & & x \notin \mathfrak{G} \quad \neg x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \cap (\bar{\mathfrak{f}} \setminus \mathfrak{G})) \text{ cl. unitaires} \\
 & & x \notin \mathfrak{F} & x \in \mathfrak{g} \quad \neg x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \cap \mathfrak{g} \setminus \mathfrak{F}) \text{ cl. unitaires} \\
 & & & x \in \mathfrak{G} \setminus \mathfrak{g} \quad \neg x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \cap \bar{\mathfrak{g}} \setminus \mathfrak{F}) \text{ cl. unitaires} \\
 & & & x \notin \mathfrak{G} \quad \neg x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \setminus \mathfrak{F} \setminus \mathfrak{G}) \text{ cl. unitaires} \\
 & x \notin \mathfrak{H} & x \in \mathfrak{f} & x \in \mathfrak{g} \quad false \\
 & & & x \in \mathfrak{G} \setminus \mathfrak{g} \quad \neg x_{\mathcal{G}} \quad \#(\mathfrak{f} \cap (\bar{\mathfrak{g}} \setminus \mathfrak{H})) \text{ cl. unitaires} \\
 & & & x \notin \mathfrak{G} \quad true \\
 & & x \in \mathfrak{F} \setminus \mathfrak{f} & x \in \mathfrak{g} \quad \neg x_{\mathcal{F}} \quad \#(\bar{\mathfrak{f}} \cap \mathfrak{g} \setminus \mathfrak{H}) \text{ cl. unitaires} \\
 & & & x \in \mathfrak{G} \setminus \mathfrak{g} \quad \neg x_{\mathcal{F}} \vee \neg x_{\mathcal{G}} \quad \#(\bar{\mathfrak{f}} \cap \bar{\mathfrak{g}} \setminus \mathfrak{H}) \text{ cl. binaires} \\
 & & & x \notin \mathfrak{G} \quad true \\
 & & x \notin \mathfrak{F} & x \in \mathfrak{g} \quad true \\
 & & & x \in \mathfrak{G} \setminus \mathfrak{g} \quad true \\
 & & & x \notin \mathfrak{G} \quad true
\end{array}
$$

TABLE 1 – Règle d'encodage pour l'intersection de deux ensembles

joue à domicile alors que la seconde joue à l'extérieur. Un tournoi doit satisfaire les 3 contraintes suivantes : toutes les équipes jouent une fois par semaine ; toutes les équipes jouent au plus deux fois sur la même période au cours du tournoi ; toutes les équipes rencontrent toutes les autres équipes. La valeur n permet de définir totalement une instance du problème. Ce problème peut être vu comme un le problème du Round Robin Tournament auquel une contrainte sur les périodes à été ajoutée.

Voici un modèle ensembliste où les $n-1$ semaines sont notées w et les $n/2$ périodes sont notées p :

- Univers et ensembles d'équipes : $\mathcal{U} :: 1..n, \mathcal{T} :: \mathcal{U}, \mathcal{U}, n, n$
- Matches de 2 équipes pour chaque semaine et chaque période : $\forall i \in [1..w], \forall j \in [1..p], \quad G_{i,j} :: \perp, \mathcal{U}, 2, 2$
- Chaque équipe joue chaque semaine : $\forall i \in [1..w], \ \mathcal{T} = \bigcup_{j=1}^{p} G_{i,j}$
- Chaque équipe joue au plus 2 fois dans une période : $\forall q \in [1..p], \ \forall i \in [1..w-2], \ \forall j \in [i+1..w-1], \forall k \in [j+1..w], \emptyset = G_{i,p} \cap G_{j,p} \cap G_{k,p}$
- Chaque équipe rencontre toutes les autres équipes : puisque chaque équipe joue chaque semaine, il est suffisant de forcer que chaque paire de matches partage au plus une équipe : $\forall i \in [1..w-1], \ \forall j \in [i+1..w], \ \forall p_1, p_2 \in [1..p], G_{i,p_1,j,p_2} :: \perp, \mathcal{U}, 0, 1 \wedge G_{i,p_1,j,p_2} = G_{i,p_1} \cup G_{j,p_2}$
- Symétrie 1. La première semaine est simplement remplie : l'équipe 1 joue contre la 2 dans la 1ère période, la 3 contre la 4 dans la 2ème, etc $\forall i \in [1..n], \quad i \in G_{1,((i-1)div2)+1}$
- Symétrie 2. La première équipe est placée pendant p semaines (en diagonale, en partant de la seconde semaine) : $\forall i \in [1..p], \quad 1 \in G_{i+1,i}$

5.2 Pré-traitement SAT

L'utilisation d'un pré-traitement est recommandé pour minimiser la taille des instances CNF. Il est aussi connu que les instances réduites ne sont pas forcément plus simples à résoudre. En effet, il est possible que des solutions facilement atteignables soient retirées par le pré-traitement et que seules les solutions plus difficiles à atteindre soient conservées. Nous utilisons SatElite [9] comme "CNF minimizer". Nous pouvons l'utiliser de manière complète (CM_{Sat}) avec subsomption, symétries, ... ou uniquement pour appliquer la propagation unitaire comme pré-traitement (UP_{Sat}). Les comparaisons sont réalisées pour les 3 problèmes présentés précédemment et les résultats apparaissent dans le tableau 2. Le modèle $unrefined$ (i.e., sans pré-traitement SAT) est obtenu après l'unique application de l'encodage $\Leftrightarrow_{enc}$. Le modèle réduit (avec $\Rightarrow_{red}$) est également calculé pour chaque instance. Le suffixe "_R" est ajouté au nom de l'instance réduite.

Le tableau 2 montre que les instances non-raffinées sont résolues plus facilement que les instances ayant subit un pré-traitement SAT.

Pour les petites instances (toutes celles du Sudoku, et les premières du SGP et STS), les règles de réduction n'améliorent pas les résultats : en effet, elles nécessitent trop de temps par rapport au gain obtenu par la suite avec Minisat. Pour toutes les instances SGP avec le modèle "implication", et les instances STS (exceptée la première), l'application des règles de réduction améliore à la fois la taille et le temps de résolution.

5.3 Résultats pour de grandes instances

La section précèdente a montré que le problème du Sudoku est très simple à résoudre. Nous nous concentrons maintenant sur des problèmes plus complexes : le SGP et STS.

Dans [18], seule la modélisation et l'encodage sans réduction a été étudié. Les instances générées étaient plus petites (variables et clauses) et résolues plus rapidement que les instances générées directement. Le tableau 3 montre que le modèle à base d'implication associé aux réductions permet une résolution plus rapide. Des solutions sont calculées pour les instances SGP 8_4_8 et 9_4_9, et à notre connaissance, c'est le premier modèle SAT qui permet de résoudre ces instances. Il est à noter que les instances 8_4_11, 8_4_12 et 9_4_12 n'ont pas de solution ; dans ce cas, le solveur doit prouver l'insatisfiabilité des instances.

Le tableau 4 montre l'impact important qu'a la réduction sur notre encodage SAT. En effet, sans réduction les instances STS sont résolues jusqu'à la taille 12 alors qu'avec réduction, des solutions sont trouvées jusqu'à la taille 66. Comme précisé en section 5.2, un procédé de réduction du modèle pourrait compliquer la résolution (par exemple en retirant solution symétriques). Nous pouvons observer que ce n'est pas le cas ici. Nos règles de réduction simplifient la recherche : elles gardent la structure du problème, sans retirer de solutions (même symétriques), et réduisent l'espace de recherche. La taille des instances n'est donc pas l'unique critère : la structure du problème et la taille de l'espace de recherche est également primordial. Par exemple, des instances plus hautes mais réduites, peuvent être plus grandes (nombres de variables et clauses) que des instance plus simples non réduites ; mais les instances réduites (bien que plus grandes en variables et clauses) peuvent être résolues alors que les non-réduites ne le sont pas. Par exemple, l'instance 14 non réduite du STS est plus petite (variables, clauses) que l'instance 20 réduite ; mais l'instance 20 réduite est résolue alors que l'instance 14 non-réduite n'y est pas. Pour les instances 68 à 72, Minisat stoppe immédiatement la résolution, le nombre de clauses et de variables étant trop grand. Des solveurs spécifiques [15] ont été réalisés pour résoudre ce problème. Les problèmes eux-mêmes sont sur-contraints par l'ajout de contraintes n'apparaissant pas dans le problème initial. Ainsi, l'instance résolue la plus grande est 70, mais la moitié des instances n'ont plus de solutions (dû aux sur-contraintes).

6 Conclusion

Nous avons présenté une technique pour l'encodage des contraintes ensembliste CSP en SAT : le processus de modélisation est effectué grâce à des contraintes ensemblistes expressives ; ces contraintes sont ensuite réduites par nos règles $\Rightarrow_{red}$ avant d'être converties automatiquement ($\Leftrightarrow_{enc}$) en variables et clauses SAT. Nous avons illustré notre approche par divers problèmes (Sudoku, le problème du Social Golfer, et le problème du Sports Tournament Scheduling) et avons obtenus de bons résultats en appliquant nos règles de réduction et d'encodage. Les avantages de notre technique sont les suivants :

- la modélisation est simple, expressive et lisible. De plus, les modèles sont indépendants des solveurs SAT ou CSP ;
- la technique est beaucoup moins sensible aux erreurs qu'un encodage direct en SAT ;
- les instances SAT générées automatiquement sont plus petites en terme de nombres de variables et clauses ;
- finalement, l'ajout d'une phase de réduction permet de réduire le temps total de résolution (réduction+encodage+résolution) ;
- les instances SAT générées semblent aussi être

TABLE 2 – Efficiency of SAT pre-processes

	Instances	Model characteristics						Encoding time			Resolving time				
		$\Rightarrow_{red}$	Unrefined		UP_{Sat}		CM_{Sat}		$\Leftrightarrow_{enc}$	UP_{Sat}	CM_{Sat}	Unrefined		UP_{Sat}		CM_{Sat}	
		sec.	#cl	#var	#cl	#var	#cl	#var	sec.	sec.	sec.	sec.	sum	sec.	sum	sec.	sum
Sudoku	empty	-	19 728	4 122	13 527	2 997	12 474	2 430	0.04	0.02	0.17	0.47	0.51	0.18	**0.24**	0.14	0.35
	empty_R	0.03	14 031	3 339	13 527	2 997	12 474	2 430	0.03	0.01	0.16	0.18	**0.25**	0.18	0.26	0.14	0.37
	523262	-	19 757	4 122	8 684	1 924	8 008	1 560	0.04	0.03	0.11	0.14	**0.18**	0.16	0.22	0.21	0.36
	523262_R	0.12	7 260	1 936	6 814	1 594	5 901	1 223	0.02	0.00	0.06	0.03	**0.20**	0.08	0.22	0.09	0.30
	527796	-	19 756	4 122	8 851	1 961	8 162	1 590	0.04	0.03	0.11	0.08	0.12	0.04	**0.10**	0.06	0.20
	527796_R	0.20	6 648	1 828	6 202	1 486	4 998	1 043	0.02	0.00	0.06	0.01	**0.24**	0.02	**0.24**	0.01	0.30
	53151	-	19 754	4 122	9 185	2 035	8 470	1 650	0.04	0.03	0.11	0.12	**0.16**	0.11	0.18	0.13	0.28
	53151_R	0.16	7 461	1 994	7 009	1 652	5 908	1 207	0.02	0.01	0.08	0.05	**0.23**	0.05	0.24	0.05	0.30
	55112	-	19 757	4 122	8 684	1 924	8 008	1 560	0.04	0.03	0.11	0.06	**0.07**	0.03	0.10	0.05	0.20
	55112_R	0.11	7 897	2 049	7 451	1 707	6 789	1 387	0.02	0.01	0.06	0.02	**0.15**	0.02	0.16	0.02	0.21
	58657	-	19 760	4 122	8 183	1 813	7 546	1 470	0.04	0.03	0.10	0.03	**0.07**	0.02	0.08	0.02	0.17
	58657_R	0.17	6 651	1 804	6 211	1 462	5 070	1 036	0.02	0.00	0.06	0.01	**0.20**	0.01	**0.20**	0.01	0.26
	128214	-	19 762	4 122	7 849	1 739	7 238	1 410	0.04	0.03	0.10	0.01	**0.05**	0.02	0.08	0.01	0.15
	128214_R	0.19	6 279	1 721	5 843	1 379	4 972	1 025	0.01	0.00	0.06	0.00	**0.20**	0.01	0.21	0.00	0.26
SGP	8_4.5.C	-	526 333	29 696	214 370	14 787	146 332	3 474	0.97	0.73	11.11	0.08	**1.05**	0.04	1.75	0.02	12.11
	8_4.5.C_R	0.92	216 751	16 361	193 582	14 060	139 919	3 109	0.43	0.18	7.72	0.05	**1.40**	0.04	1.57	0.02	9.09
	8_4.6.C	-	745 338	41 760	337 665	22 684	230 357	4 346	1.38	1.00	21.68	0.12	**1.50**	0.08	2.46	0.04	23.10
	8_4.6.C_R	1.19	334 591	24 422	305 647	21 570	220 847	3 889	0.67	0.29	15.18	0.09	**1.94**	0.07	2.22	0.04	17.08
	8_4.5.I	-	464 893	9 216	191 410	3 811	176 004	3 601	3.13	5.16	5.91	0.06	**3.19**	0.02	8.31	0.03	9.07
	8_4.5.I_R	-	193 987	5 861	170 858	3 600	168 994	3 220	1.70	0.26	0.75	0.06	**1.76**	0.02	1.98	0.03	2.48
	8_4.6.I	-	653 178	11 040	300 145	4 764	276 245	4 503	4.64	8.80	10.52	0.13	**4.77**	0.07	13.51	0.06	15.22
	8_4.6.I_R	-	297 921	7 292	269 037	4 500	266 414	4 024	2.67	0.38	1.31	0.04	**2.71**	0.05	3.09	0.05	4.03
STS	8	-	23 812	5 528	13 794	3 867	6 422	1 040	0.04	0.04	0.28	0.01	**0.05**	0.01	0.09	0.01	0.33
	8_R	0.08	16 606	4 549	13 345	3 902	2 649	426	0.03	0.02	0.26	0.00	**0.11**	0.00	0.13	0.00	0.37
	10	-	77 135	17 170	46 917	12 530	29 233	5 194	0.14	0.12	0.91	3.37	3.51	1.18	**1.44**	5.32	6.37
	10_R	0.11	58 235	14 788	45 918	12 650	13 729	2 352	0.11	0.07	0.74	0.01	**0.23**	0.01	0.30	-	0.97
	12	-	198 198	42 612	124 980	32 297	85 581	15 576	0.35	0.40	3.08	143.96	**144.31**	509.24	509.99	266.26	269.70
	12_R	0.22	148 364	35 188	112 467	29 541	40 188	6 624	0.39	0.22	2.22	0.04	**0.65**	0.03	0.86	0.01	2.84

TABLE 3 – Results for large SGP instances using cardinality and implication models with reduction rules.

Inst.	cardinality model						implication model					
	$\Rightarrow_{red}$	Unrefined		$\Leftrightarrow_{enc}$	Resolution		$\Rightarrow_{red}$	Unrefined		$\Leftrightarrow_{enc}$	Resolution	
	sec.	#cl	#var	sec.	sec.	sum	sec.	#cl	#var	sec.	sec.	sum
8_4_5	0.92	216 751	16 361	0.03	0.05	1.00	0.03	193 987	5 861	0.01	0.03	**0.07**
8_4_6	1.19	334 591	24 422	0.04	0.15	1.37	0.31	297 921	7 292	0.21	0.10	0.62
8_4_7	1.56	477 865	34 085	0.06	0.71	2.33	0.06	424 003	8 723	0.03	0.16	**0.25**
8_4_8	1.95	646 573	45 350	0.09	-	-	0.08	572 233	10 154	0.07	0.26	**0.41**
8_4_9	2.46	840 715	58 217	0.13	-	-	0.12	742 611	11 585	0.05	-	-
8_4_10	2.93	1 060 291	72 686	0.15	-	-	0.18	935 137	13 016	0.08	-	-
8_4_11	3.45	1 305 301	88 757	0.18	-	-	0.21	1 149 811	14 447	0.13	-	-
8_4_12	4.10	1 575 745	106 430	0.2	-	-	0.26	1 386 633	15 878	0.08	-	-
9_4_6	1.58	523 471	34 079	0.07	0.17	1.82	0.04	468 991	9 489	0.03	0.09	**0.16**
9_4_7	2.09	750 568	47 818	0.09	0.23	2.42	0.12	670 306	11 356	0.07	0.26	**0.45**
9_4_8	2.65	1 018 441	63 875	0.13	1.37	4.15	0.11	907 435	13 223	0.09	0.38	**0.58**
9_4_9	3.24	1 327 090	82 250	0.17	-	-	0.23	1 180 378	15 090	0.09	6.37	**6.69**
9_4_10	3.90	1 676 515	102 943	0.22	-	-	0.26	1 489 135	16 957	0.08	-	-
9_4_11	4.79	2 066 716	125 954	0.27	-	-	0.31	1 833 706	18 824	0.20	-	-
9_4_12	5.43	2 497 693	151 283	0.33	-	-	0.33	2 214 091	20 691	0.18	-	-

TABLE 4 – Résultats pour de grandes instances de STS

Inst.	Initial					reduced					
	Unrefined		$\Leftrightarrow_{enc}$	Resolution		$\Rightarrow_{red}$	Unrefined		$\Leftrightarrow_{enc}$	Resolution	
	#cl $\times 10^3$	#var$\times 10^3$	sec.	sec.	sum	sec.	#cl$\times 10^3$	#var$\times 10^3$	sec.	sec.	sum
8	24	5 528	0.02	0.04	**0.06**	0.08	15	4	0.02	0.02	0.12
10	77	17 170	0.01	9.30	9.31	0.11	54	13	0.01	0.04	**0.16**
12	198	43	0.03	387.85	387.88	0.22	148	35	0.02	0.14	**0.38**
14	437	91	0.04	-	-	0.41	343	78	0.03	0.22	**0.66**
16	861	175	0.09	-	-	0.66	700	154	0.05	0.45	**1.16**
18	1 569	314	0.17	-	-	0.98	1 307	281	0.14	0.89	**2.01**
20	2 676	528	0.32	-	-	1.34	2 275	479	0.24	1.69	**3.28**
22	4 331	842	0.49	-	-	1.92	3 742	773	0.41	3.43	**5.76**
24	6 713	1 289	0.75	-	-	2.65	5 879	1 194	0.66	4.82	**8.13**
26	10 041	1 905	1.15	-	-	3.71	8 890	1 779	0.98	9.24	**13.93**
28	14 567	2 735	1.62	-	-	5.34	13 020	2 569	1.37	12.37	**19.08**
30	20 591	3 827	2.26	-	-	6.13	18 551	3 616	2.03	20.20	**28.36**
32	28 453	5 241	3.18	-	-	8.11	25 813	4 974	2.92	31.93	**42.96**
34	38 581	7 059	4.00	-	-	10.62	35 203	6 719	3.89	36.17	**50.68**
36	51 396	9 343	5.54	-	-	12.67	47 151	8 925	5.04	83.38	**101.09**
38	67 397	12 178	7.26	-	-	14.96	62 129	11 668	6.64	124.25	**145.85**
40	87 144	15 655	9.29	-	-	12.78	80 678	15 041	8.44	114.74	**135.96**
50	266 070	46 637	27.93	-	-	36.93	250 325	45 254	27.38	880.50	**944.81**
52	323 676	56 497	33.62	-	-	46.99	305 266	54 901	31.79	1240.79	**1319.57**
54	390 835	67 948	40.61	-	-	69.64	369 437	66 116	39.13	1492.06	**1600.83**
56	468 687	81 175	48.36	-	-	60.58	443 952	79 083	46.24	1803.75	**1910.57**
58	558 459	96 375	67.49	-	-	60.19	530 012	93 996	55.61	2362.21	**2478.01**
60	661 467	113 760	70.66	-	-	78.28	628 906	111 067	64.76	2952.74	**3095.78**
62	779 123	133 556	79.58	-	-	87.63	742 017	130 519	77.08	3367.36	**3532.07**
64	912 933	156 005	95.06	-	-	100.48	870 823	152 592	92.14	4903.03	**5095.65**
66	1 064 779	181 500	109.51	-	-	115.36	1 017 067	177 625	105.45	5247.73	**5468.54**
68	1 236 149	210 203	126.70	-	-	135.61	1 182 405	205 875	122.12	-	-
70	1 428 864	242 406	145.71	-	-	154.44	1 368 535	237 589	139.81	-	-
72	1 644 854	278 418	172.92	-	-	182.35	1 577 355	273 071	167.62	-	-

bien adaptées à Minisat.

Nous prévoyons d'utiliser de combiner nos contraintes ensemblistes avec l'arithmétique sur les domaines finis, et de fournir également l'encodage SAT de ces nouvelles contraintes. A ces fins, nous devrons ajouter de nouvelles contraintes et compléter nos ensembles de règles $\Leftrightarrow_{enc}$ et $\Rightarrow_{red}$. Nous avons notre propre format pour nos modèles (XML-like) mais nous prévoyons de passer au standard XCSP3 [7].

Références

[1] CHOCO. http ://www.emn.fr/z-info/choco-solver/.

[2] K. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[3] Francisco Azevedo. Cardinal : A finite sets constraint solver. *Constraints*, 12(1) :93–129, 2007.

[4] F. Bacchus. Gac via unit propagation. In *Proc. of CP 2007*, volume 4741 of *LNCS*, pages 133–147. Springer, 2007.

[5] O. Bailleux and Y. Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *Proc. of CP 2003*, volume 2833, pages 108–122. Springer, 2003.

[6] C. Bessière, E. Hebrard, and T. Walsh. Local consistencies in sat. In *Selected Revised Papers of SAT 2003.*, volume 2919 of *LNCS*, pages 299–314. Springer, 2004.

[7] Frédéric Boussemart, Christophe Lecoutre, Cédric Piette, and Vincent Perradin. XCSP3 an integrated format for benchmarking combinatorial constrained problems. http ://www.xcsp.org/.

[8] Francisco de Moura e Castro Ascensão de Azevedo. *Constraint Solving over Multi-valued Logics - Application to Digital Circuits*. PhD thesis, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2002.

[9] N. Eén and A. Biere. Effective preprocessing in sat through variable and clause elimination. In *SAT 2005*, volume 3569, pages 61–75, 2005.

[10] N. Eén and N. Sörensson. An extensible sat-solver. In *SAT 2003*, volume 2919, pages 502–518, 2003.

[11] T. Früwirth. *Constraint Handling Rules*. Cambridge University Press, 2009.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco, 1979.

[13] I. Gent and I. Lynce. A sat encoding for the social golfer problem. In *IJCAI'05 workshop on modelling and solving problems with constraints*, 2005.

[14] Carmen Gervet. Conjunto : Constraint propagation over set constraints with finite set domain variables. In *Proc. of ICLP'94*, page 733. MIT Press, 1994.

[15] Jean-Philippe Hamiez and Jin-Kao Hao. A note on a sports league scheduling problem. *CoRR*, abs/1410.2721, 2014.

[16] Warwick Harvey. CSPLib problem 010 : Social golfers problem. http ://www.csplib.org/Problems/prob010.

[17] F. Lardeux, E. Monfroy, F. Saubion, B. Crawford, and C. Castro. Sat encoding and csp reduction for interconnected alldiff constraints. In *Proc. of MICAI 2009*, pages 360–371, 2009.

[18] Frédéric Lardeux, Eric Monfroy, Broderick Crawford, and Ricardo Soto. Set constraint model and automated encoding into SAT : application to the social golfer problem. *Annals OR*, 235(1) :423–452, 2015.

[19] B. Legeard and E. Legros. Short overview of the clps system. In *Proc. of PLILP'91*, volume 528, pages 431–433. Springer, 1991.

[20] A. Mackworth. *Encyclopedia on Artificial Intelligence*, chapter Constraint Satisfaction. John Wiley, 1987.

[21] F. Rossi, T. P. van Beek, and Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

[22] M. Triska and N. Musliu. An improved sat formulation for the social golfer problem. *Annals of Operations Research*, 194(1) :427–438, 2012.

[23] Toby Walsh. CSPLib problem 026 : Sports tournament scheduling. http ://www.csplib.org/Problems/prob026.

Étude de stratégies parallèles de coopération avec POSL

Alejandro REYES-AMARO[1] **Éric MONFROY**[1] **Florian RICHOUX**[1]

[1] LINA Inria–TASC, Université de Nantes, Nantes, France
{alejandro.reyes,eric.monfroy,florian.richoux}@univ-nantes.fr

Résumé

La technologie multi-cœur et les architectures massivement parallèles sont de plus en plus accessibles à tous, à travers des matériaux comme le Xeon Phi ou les cartes GPU. Cette stratégie d'architecture a été communément adoptée par les producteurs pour faire face à la loi de Moore. Or, ces nouvelles architectures impliquent d'autres manières de concevoir et d'implémenter les algorithmes, pour exploiter complètement leur potentiel, en particulier dans le cas des solveurs de contraintes traitant de problèmes d'optimisation combinatoire. Dans cet article on utilise un Langage pour créer des Solveurs Orienté Parallèle (POSL pour Parallel-Oriented Solver Language) : cadre permettant de construire des solveurs basés sur des méta-heuristiques interconnectées travaillant en parallèle, dans le but de résoudre des instances des problèmes *Social Golfers* et *Costas Array* et de mesurer sa performance. Nous testons plusieurs stratégies de résolution, grâce au langage orienté parallèle, basé sur des opérateurs, que POSL fournis.

Abstract

The multi-core technology and massive parallel architectures are nowadays more accessible for a broad public through hardware like the Xeon Phi or GPU cards. This architecture strategy has been commonly adopted by processor manufacturers to stick with Moore's law. However, this new architecture implies new ways to design and implement algorithms to exploit its full potential. This is in particular true for constraint-based solvers dealing with combinatorial optimization problems. In this paper we use Parallel-Oriented Solver Language (POSL), a framework to build interconnected meta-heuristic-based solvers working in parallel, by using communications operators, to solve instances of *Social Golfers* and *Costas Array* problems and measure its performance. We test many different solution's strategies, thanks to a parallel-oriented language provided, based on operators.

1 Introduction

L'optimisation combinatoire a plusieurs applications dans différents domaines tels que l'apprentissage de la machine, l'intelligence artificielle, et le génie du logiciel. Dans certains cas, le but principal est seulement de trouver une solution, comme pour les Problèmes de Satisfaction de Contraintes (CSP). Une solution sera une affectation de variables répondant aux contraintes fixées, en d'autres termes : une solution faisable.

Les *CSP*s sont connus pour être des problèmes extrêmement difficiles. Parfois les méthodes complètes ne sont pas capables de passer à l'échelle de problèmes de taille industriel. C'est la raison pour laquelle les techniques méta–heuristiques sont de plus en plus utilisées pour la résolution de ces derniers. Par contre, dans la plupart des cas industriels, l'espace de recherche est assez important et devient donc intraitable, même pour les méthodes méta-heuristiques. Cependant, les récents progrès dans l'architecture de l'ordinateur nous conduisent vers les ordinateurs *multi/many–cœur*, en proposant une nouvelle façon de trouver des solutions à ces problèmes d'une manière plus réaliste, ce qui réduit le temps de recherche.

Grâce à ces développements, les algorithmes parallèles ont ouvert de nouvelles façons de résoudre les problèmes de contraintes : Adaptive Search [5] est un algorithme efficace, montrant de très bonnes performances et passant à l'échelle de plusieurs centaines ou même milliers de cœurs, en utilisant la recherche locale *multi-walk* en parallèle. Munera et al. [10] ont présenté une autre implémentation d'Adaptive Search en utilisant la coopération entre des stratégies de recherche. Ces travaux ont montré l'efficacité du schéma parallèle multi-walk.

De plus, le temps de développement nécessaire pour coder des solveurs en parallèle est souvent sous-estimé, et dessiner des algorithmes efficaces pour résoudre certains problèmes consomment trop de temps. Dans [13] nous pré-

sentons POSL, un langage orienté parallèle pour construire des solveurs de contraintes basés sur des méta-heuristiques, qui résolvent des *CSP*s. Il fournit un mécanisme pour coder des stratégies de communication indépendantes du Le but de cet article est de proposer des nouveaux opérateurs de communication, très utiles pour dessiner des stratégies de communication, et de présenter une analyse détaillée des résultats obtenus en résolvant plusieurs instances des problèmes *Social Golfers* et *Costas Array*. Sachant que créer des solveurs utilisant différentes stratégies de solution peut être compliqué et pénible, POSL donne la possibilité de faire des prototypes de solveurs communicants facilement.

2 Des travaux reliés

Beaucoup de chercheurs se concentrent sur la programmation par contraintes, particulièrement dans le développement de solution à haut-niveau qui facilitent la construction de stratégies de recherche. Cela permet de citer plusieurs contributions.

HYPERION [3] est un système codé en Java pour méta et hyper-heuristiques basé sur le principe d'interopérabilité, fournissant des patrons génériques pour une variété d'algorithmes de recherche locale et évolutionnaire, et permettant des prototypages rapides avec la possibilité de réutiliser le code source. POSL offre ces avantages, mais il fournit également un mécanisme permettant de définir des protocoles de communication entre solveurs. Il fournit aussi, à travers d'un simple langage basé sur des opérateurs, un moyen de construire des *computation strategies*, en combinant des *composants* déjà définis (*operation modules* et *open channels*). Une idée similaire a été proposée dans [6] sans communication, qui introduit une approche évolutive en utilisant une simple composition d'opérateurs pour découvrir automatiquement les nouvelles heuristiques de recherche locale pour SAT et les visualiser comme des combinaisons d'un ensemble de blocs.

Récemment, [15] a montré l'efficacité de combiner différentes techniques pour résoudre un problème donné (hybridation). Pour cette raison, lorsque les composant du solveurs peuvent être combinés, POSL est dessiné pour exécuter en parallèle des ensembles de solveurs différents, avec ou sans communication. Une autre idée intéressante est proposée dans TEMPLAR. Il s'agit d'un système qui génère des algorithmes en changeant des composants prédéfinis, et en utilisant des méthodes hyper-heuristiques [14]. Dans la dernière phase du processus de codage avec POSL, les solveurs peuvent être connectés les uns aux autres, en fonction de la structure de leurs *open channels*, et de cette façon, ils peuvent partager non seulement des informations, mais aussi leur comportement, en partageant leurs *operation modules*. Cette approche donne aux solveurs la capacité d'évoluer au cours de l'exécution.

Renaud De Landtsheer et al. présentent dans [7] un cadre facilitant le développement des systèmes de recherches en utilisant des *combinators* pour dessiner les caractéristiques trouvées très souvent dans les procédures de recherches comme des briques, et les assembler. Dans [9] est proposée une approche qui utilise des systèmes coopératifs de recherche locale basée sur des méta-heuristiques. Celle-ci se sert de protocoles de transfert de messages. POSL combine ces deux idées pour assembler des composants de recherche locale à travers des opérateurs fournis (ou en créant des nouveaux), mais il fournit aussi un mécanisme basé sur opérateurs pour connecter et combiner des solveurs, en créant des stratégies de communication.

Une présentation plus détaillée de POSL est disponible dans [11,12], où le lecteur peut trouver une description formelle sur la façon d'utiliser le langage basé sur des opérateurs pour construire des prototypes de solveurs. Dans cet article, nous présentons quelques nouveaux opérateurs de communication afin de concevoir des stratégies de communication. Avant de clore cet article par une brève conclusion et de travaux futurs, nous présentons quelques résultats obtenus en utilisant POSL pour résoudre certaines instances des problèmes *Social Golfers* et *Costas Array*.

3 Construire des solveurs en parallèle avec POSL

Dans cette section, nous présentons un résumé des étapes à suivre pour construire des solveurs parallèles communicatifs en utilisant POSL. L'idée principale est de combiner des modules disponibles dans ce cadre, ou d'en créer des nouveaux, et de les coller à travers POSL pour créer des solveurs indépendants. Après, nous pouvons les connecter en utilisant des *opérateurs de connexion*. Nous appelons l'entité finale obtenue POSL **Meta-Solver**.

3.1 PREMIÈRE ÉTAPE : Créer les modules de POSL

Il existe deux types de modules de base dans POSL : les *operation modules* et les *open channel*. Un *operation module* est une fonction qui reçoit une entrée, puis il exécute un algorithme interne et renvoie une sortie. Les types d'entrée et de sortie définiront l'*operation module*. Il peut être remplacé dynamiquement ou combiné avec d'autres *operation modules*, car ils peuvent être partagés entre les solveurs travaillant en parallèle. De cette manière, le programme de calcul peut changer son comportement au cours de l'exécution. La figure 1 montre un exemple d'*operation module* : il reçoit une configuration S puis calcule l'ensemble V de ses configurations voisines $\{S^1, S^2, \ldots, S\}$.

Un *open channel* est également une fonction qui reçoit et renvoie des informations, mais contrairement à l'*operation module*, l'*open channel* peut recevoir des informations de deux façons différentes : par le paramètre ou de l'extérieur,

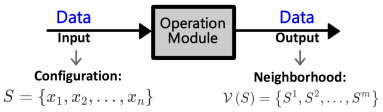
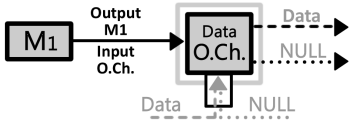
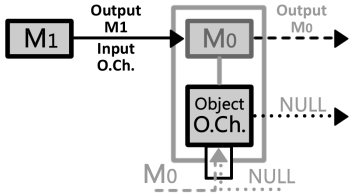


FIGURE 1 – Un *operation module*



(a) Data *open channel*



(b) Object *open channel*

FIGURE 2 – Les *open channels*

c'est-à-dire en communiquant avec un module d'un autre solveur. L'*open channel* est le composant responsable de la réception de l'information dans les communications entre les solveurs. Il peut interagir avec les *operation modules* grâce aux opérateurs.

Un *open channel* peut recevoir deux types d'informations, toujours en provenance d'un solveur externe : des données ou des *operation modules*. Il est important de remarquer que lorsque nous parlons de l'envoi–réception d'*operation modules*, nous entendons l'envoi–réception d'information seulement nécessaire pour les identifier et pouvoir les instancier.

Afin de distinguer les deux types d'*open channels*, nous appellerons Data Open Channel celui qui est responsable de la réception de données (figure 2a), et Object Open Channel de la réception et l'instanciation d'*operation modules* (figure 2b).

Les utilisateurs de POSL peuvent implémenter des nouveaux modules (*operation modules* et *open channels*) mais POSL contient déjà plusieurs modules très utiles pour résoudre un large éventail de problèmes.

3.2 DEUXIÈME ÉTAPE : **Assembler des modules de** POSL

Dans cette étape, une stratégie générique est codée par POSL, en utilisant les modules mentionnés dans la première étape. Elle permet non seulement l'échange d'information, mais aussi l'exécution de modules en parallèle. Nous appelons cela la *computation strategy*.

La *computation strategy* est le cœur du solveur. C'est un programme qui joint les *operation modules* et *open channels* d'une manière cohérente, en restant indépendant des ces derniers. Grâce à la *computation strategy* nous pouvons aussi choisir les informations qui seront envoyées à d'autres solveurs. Une présentation plus formelle de la spécification des opérateurs de POSL est disponible dans [11, 12].

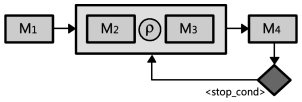


FIGURE 3

La figure 3 présente un exemple simple de la façon de combiner des *modules* en utilisant les opérateurs mentionnés précédemment, et l'algorithme 1 [1] est le pseudo-code POSL correspondant. Dans cet exemple nous montrons quatre *operation modules* qui font partie d'un *compound module* représentant une méthode trivial de recherche locale.

- `M1` : produit une configuration au hasard,
- `M2` : calcule le voisinage d'une configuration donnée, en sélectionnant une variable au hasard, et en changeant sa valeur,
- `M3` : calcule le voisinage d'une configuration donnée, en sélectionnant K variables au hasard, et en changeant leur valeur,
- `M4` : sélectionne (et sauvegarde), dans un ensemble de configurations, celle qui a le coût le plus petit.

Dans cet exemple :

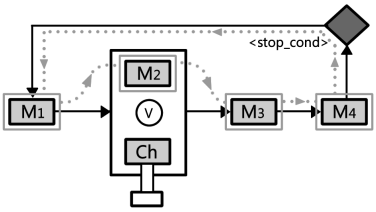
- La fonction **execute** `(A,B)` exécute le module `A` puis le module `B`
- La fonction **while** exécute le module tant qu'une condition donnée est vraie. Dans l'exemple ci-dessous, le processus est répété `N` fois.
- L'*operation module* `M2` est exécuté avec une probabilité ρ (`p` dans le pseudo-code), et `M3` est exécuté avec une probabilité $(1-\rho)$, en utilisant la fonction (opérateur) **rho**.

Algorithm 1: POSL pseudo-code pour la figure 3

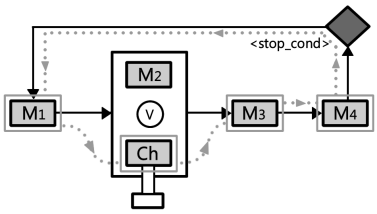
```
1 execute(M1,  while(loops < N)
2                 execute( rho(p, M2, M3), M4);
3              end
4 );
```

La figure 4 montre la manière de combiner un *open channel* avec l'*operation module* `M2` en utilisant l'opérateur (∨). Dans ce cas, l'*operation module* `M2` est exécuté tant que l'*open channel* est *NULL*, c'est-à-dire qu'il n'y a

1. Il est nécessaire de signaler que ce pseudo-code est uniquement utilisé pour faciliter la compréhension du principe du POSL. Des exemples du vrai code POSL son disponibles sur https ://github.com/alejandro-reyesamaro/POSL



(a) Le solveur exécute son propre *operation module* si aucune information n'a été reçue par l'*open channel*



(b) Le solveur exécute l'*operation module* qui est arrivé par l'*open channel*

FIGURE 4 – Deux comportements différents du même solveur

pas d'information provenant de l'extérieur. Le comportement est représenté dans la figure 4a par les lignes pointillées. Si un *operation module* a été reçu par l'*open channel*, il est exécuté au lieu de l'*operation module*() `M2`, et ce comportement est représenté dans la figure 4b par des lignes pointillées.

En utilisant ces opérateurs, nous pouvons créer l'algorithme en manipulant différents composants pour trouver la solution d'un problème donné. Ces algorithmes sont fixes mais génériques par rapport à leurs composants (*operation modules* et *open channels*). Cela signifie que nous pouvons construire différents solveurs en utilisant la même stratégie (*computation strategy*), mais avec l'instanciation de différents composants, s'ils ont la bonne signature d'entrée-sortie.

Pour définir une *computation strategy*, nous utilisons l'environnement présenté dans l'algorithme 2, où `Mi` et `Chi` représentent les types des *operation modules* et les types des *open channels* utilisés par la *computation strategy* `St`, déclarée en utilisant le mot-clé **strategy**. Entre accolades, le champ `<computation strategy>` correspond au code POSL basé sur les opérateurs en combinant des *compound modules* déjà déclarés.

Algorithm 2: Définition de la *computation strategy*

```
1 St := strategy
2    oModules: M1, M2, ..., Mn;
3    oChannels: Ch1, Ch2, ..., Chm;
4 {
5    <computation strategy>
6 }
```

Algorithm 3: Définition du solveur

```
1 Sk := solver
2 {
3    cStrategy: st;
4    oModules: m1, m2, ..., mn;
5    oChannels: ch1, ch2, ..., chm;
6 }
```

3.3 TROISIÈME ÉTAPE : **Créer les solveurs**

Avec les *operation modules* et les *open channels* déjà assemblés par la *computation strategy*, nous pouvons créer des solveurs en instanciant les composants déclarés. POSL fournit un environnement à cette fin, présenté dans l'algorithme 3, où `mi` et `chi` représentent les instances des *operation modules* et celles des *open channels* qui sont passés par des paramètres à la *computation strategy* `St`. Cela nous permet de créer de nombreux différents solveurs partageant la même *computation strategy*, mais en exécutant différents *operation modules*. Dans le pseudo-code nous utilisons le mot-clé **solver** pour signaler que nous déclarons un solveur.

3.4 QUATRIÈME ÉTAPE : **Connecter les solveurs**

Une fois que nous avons défini notre stratégie de solveur, la dernière étape consiste à déclarer les *communication channels*, c'est-à-dire connecter les solveurs les uns aux autres. Jusqu'ici, les solveurs sont déconnectés, mais ils ont tout pour établir la communication. Dans cette dernière étape, POSL fournit à l'utilisateur une plateforme pour définir facilement des *méta-stratégies* coopératives que les solveurs doivent suivre.

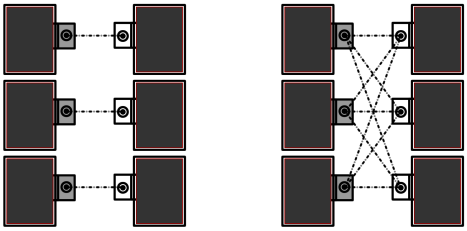
La communication est établie en suivant les règles suivantes :

1. À chaque fois qu'un solveur envoie une information via les opérateurs $(\!|.|\!)^o$ ou $(\!|.|\!)^m$, il crée une *prise mâle de communication*,
2. À chaque fois qu'un solveur contient un *open channel*, il crée une *prise femelle de communication*,
3. Les solveurs peuvent être connectés entre eux en créant des *communication channels*, reliant des prises mâles et femelles.

Avec l'opérateur $(\cdot)$ nous pouvons avoir accès aux noms des *open channels* d'un solveur, et aux *operation modules* qui envoient des informations. Par exemple : $Solver_0 \cdot M_0$ fournit un accès à l'*operation module* M_0 de $Solver_0$ si et seulement si il est utilisé par l'opérateur $(\!|.|\!)^o$ (ou $(\!|.|\!)^m$), et $Solver_1 \cdot Ch_1$ fournit un accès à l'*open channel* Ch_1 de $Solver_1$.

Donc, si nous définissons deux ensembles de solveurs, nous pouvons les connecter de deux manières différentes :

1. En connectant chaque solveur du premier ensemble, avec un solveur dans l'autre ensemble (un à un) (voir figure 5a)



(a) 1 à 1 (b) 1 à N

FIGURE 5 – Opérateurs de communication

2. En connectant chaque solveur du premier ensemble, avec tous les solveurs dans l'autre ensemble (un à N) (voir figure 5b)

Ces opérateurs i) affectent (programment) une unité disponible de calcul (de processus ou de fil, selon l'architecture utilisée) à chaque solveur impliqué, et ii) connectent les solveurs impliqués les uns aux autres.

Les opérateurs définis ci-dessus n'expriment que la possibilité de définir statiquement des stratégies de communication. À l'avenir, nous aimerions améliorer POSL en incluant des opérateurs qui permettraient plus de souplesse et d'expressivité en terme de communication entre solveurs, notamment à travers des stratégies dynamiques de communication.

4 Concevoir des expériences

Le but principal de cet article est de sélectionner deux problèmes de référence, *Social Golfers* et le *Costas Array*, pour analyser et illustrer la versatilité de POSL pour étudier des stratégies de solution basées sur la recherche locale méta-heuristique. Grâce à POSL nous pouvons analyser des résultats et formuler des conclusions sur le comportement de la stratégie de recherche, mais aussi sur la structure de l'espace de recherche du problème. Dans cette section, nous expliquons la structure des solveurs de POSL [2] que nous avons générés pour les expériences.

Le problème de *Social Golfers* consiste à planifier $n = g \times p$ golfeurs en g groupes de p joueurs chaque semaine pendant w semaines, de telle manière que deux joueurs jouent dans le même groupe au plus une fois. Une instance de ce problème peut être représentée par le triplet $g-p-w$. Ce problème, et d'autres problèmes étroitement liés, trouvent de nombreuses applications pratiques telles que le codage, le cryptage et les problèmes couvrants [8]. Sa structure nous a semblé intéressante car elle est similaire à d'autres problèmes, comme *Kirkman's Schoolgirl* et *Steiner Triple System*, donc nous pouvons construire des modules efficaces pour résoudre un grand éventail de problèmes.

2. Le code source de POSL est disponible sur https ://github.com/alejandro-reyesamaro/POSL

Le problème *Costas Array* consiste à trouver une matrice *Costas Array*, qui est une grille de $n \times n$ contenant n marques avec exactement une marque par ligne et par colonne et les $n(n-1)/2$ vecteurs reliant chaque couple de marques de cette grille dooivent tous être différents. Ceci est un problème très complexe trouvant une application utile dans certains domaines comme le sonar et l'ingénierie de radar, et présente de nombreux problèmes mathématiques ouverts. Ce problème a aussi une caractéristique intéressante : même si son espace de recherche grandit factoriellement, à partir de l'ordre 17 le nombre de solutions diminue drastiquement.

Nous avons choisi l'une des méthodes de solutions les plus classique pour des problèmes combinatoires : l'algorithme méta-heuristique de recherche locale. Ces algorithmes ont une structure commune : ils commencent par l'initialisation des structures de données. Ensuite, une configuration initiale s est générée. Après cela, une nouvelle configuration s' est sélectionnée dans le voisinage $V(s)$. Si s' est une solution pour le problème P, alors le processus s'arrête, et s' est renvoyée. Dans le cas contraire, les structures de données sont mises à jour, et s' est acceptée, ou non, pour l'itération suivante, en fonction de certains critères (par exemple, en pénalisant les caractéristiques des optimums locaux) [2].

4.1 Résoudre le problème *Social Golfers*

Afin de résoudre des instances du problème *Social Golfers*, nous utilisons les *operation modules* suivants fournis par POSL :

1. *operation module* de génération initiale :

   M_S génère une configuration aléatoire s, en respectant la structure du problème, c'est-à-dire que la configuration est un ensemble de w permutations du vecteur $[1..n]$.
2. *operation modules* de voisinage :

   M_V^{Std} définit le voisinage $V(s)$ permutant les joueurs parmi les groupes.

   M_V^{AS} définit le voisinage $V(s)$ permutant le joueur qui a contribué le plus au coût, avec d'autres joueurs dans la même semaine.
3. *operation modules* de sélection :

   $M_{\hat{S}}^{First}$ sélectionne la première configuration $\hat{s} \in V(s)$ qui améliore le coût actuel

   $M_{\hat{S}}^{Best}$ sélectionne la meilleure configuration $\hat{s} \in V(s)$ qui améliore le coût actuel

   $M_{\hat{S}}^{Rand}$ sélectionne une configuration aléatoire $\hat{s} \in V(s)$.
4. *operation module* d'acceptation :

   M_D évalue un critère d'acceptation pour $\hat{s}$. Dans tous les cas, le critère d'acceptation est toujours de choisir la configuration avec le moindre coût.

Grâce aux potentiel de POSL, nous avons pu tester de nombreuses stratégies en très peu de temps. Une première expérience a été réalisée pour sélectionner la meilleure fonction de voisinage pour résoudre le problème, en comparant une stratégie basique qui utilise l'*operation module* M_V^{Std}, avec une stratégie basique qui utilise l'*operation module* de voisinage M_V^{AS} basé sur l'algorithme d'*Adaptive Search* ainsi qu'avec les combinaisons de M_V^{Std} et M_V^{AS} après avoir appliquer les opérateurs (ρ) et $(\cup)$. Les algorithmes 4, 5 et 6 représentent respectivement les *computation strategies* pour chaque cas.

Les stratégies mentionnées ci-dessus ne sont pas capables de résoudre les problèmes avec plus de 3 semaines, nous avons donc implémenté une autre stratégie décrite dans l'algorithme 7. Ce dernier combine les *operation modules* de sélection $M_{s'}^{First}$ avec $M_{s'}^{Rand}$, et il tente d'améliorer le coût un certain nombre de fois. Si cela n'est pas possible, il sélectionne un voisin aléatoire pour l'itération suivante. Cette stratégie utilise l'*operation module* M_V^{AS}, et nous avons comparé deux *operation modules* de sélection : $M_{s'}^{First}$ et $M_{s'}^{Best}$.

Après cela, nous avons choisi la meilleure stratégie pour construire des solveurs communicatifs (algorithmes 8 et 9) pour comparer leurs performances avec les stratégies non communicatives. En utilisant les opérateurs de communication, nous concevons différentes stratégies :

— *Stratégie complète de communication* : tous les serveurs sont connectés (soit *1 à 1*, soit *1 à N*)

— *Stratégie de communication hybride* : un certain pourcentage des solveurs sont connectés et le reste sont des solveurs non communicatifs.

4.2 Résoudre le problème *Costas Array*

Pour étudier le problème *Costas Array*, nous avons réutilisé certains *operation modules* utilisés dans la résolution du problème *Social Golfers* : les *operation modules* de *Sélection* et d'*Acceptation*. Les nouveaux modules sont :

1. *operation module* de génération initiale :

   M_S génère une configuration aléatoire s, comme une permutation du vecteur $[1..n]$.

2. *operation module* de voisinage :

   M_V^{AS} définit le voisinage $V(s)$ permutant la variable qui a contribué le plus au coût, avec d'autres.

Nous ajoutons aussi un module de *Reset* (M_R) [3]. La stratégie de base que nous utilisons pour résoudre ce problème est présentée dans l'algorithme 10. Nous le prenons comme base pour construire l'ensemble des différentes stratégies de communication, en combinant des solveurs communicatifs et non communicatifs. En utilisant un solveur émetteur basé sur la stratégie de l'algorithme 11, nous avons testé la communication d'une bonne configuration après avoir sélectionné la configuration pour la prochaine itération (algorithme 12). Nous avons également testé la communication d'une bonne configuration pour la recevoir au moment où le solveur calcule le *reset*, (algorithmes 13 et 14).

Instance	T	It.	% réussi
5-3-7	8.13	17,347	100.00
8-4-7	16.92	7,829	100.00
9-4-8	79.60	20,779	94.28
11-7-5	3.37	664	100.00

TABLEAU 1 – Expériences avec *Social Golfers* (séquentiel)

STRATÉGIE	T	It.
Adaptive Search (AS)	**1.06**	352
Std (ρ) AS	41.53	147
Std $(\cup)$ AS	59.65	198
Standard (Std)	87.90	146

TABLEAU 2 – Expériences avec *Social Golfers* 10-10-3

5 Analyse des résultats

Les expériences [4] [5] ont été effectuées sur un processeur Intel ® Xeon ™ E5-2680 v2, 10 × 4 cœurs, 2.80GHz. Les résultats montrés dans cette section sont les moyennes de 30 runs pour chaque configuration. Dans les tableaux de résultats, les colonnes marquées **T** correspondent au temps de l'exécution en secondes et les colonnes marquées **It.** correspondent au nombre d'itérations.

Toutes les expériences de cette section sont basées sur différentes stratégies en parallèle. Nous utilisons 40 cœurs pour le problème *Social Golfers* et seulement 20 pour *Costas Array*, la machine n'étant pas complètement disponible au moment des expériences. Nous présentons dans les tableaux 1 et 7 les résultats du lancement des POSL métasolveurs pour résoudre chaque cas de manière séquentielle. Le tableau 1 montre de moyennes de temps et d'itérations beaucoup plus élevées que les moyennes que on peut trouver dans le tableau 3 colonne $OM_{s'}$ *First Improvement* (sans communication), et les résultats avec communication (tableaux 4, 5 et 6). Le même effet est visible dans le tableau 7. Ceci montre comment l'approche parallèle augmente la probabilité de trouver la solution dans un délai raisonnable (en secondes), par rapport au schéma séquentiel [1]. La colonne marquée **% success** indique le pourcentage de solveurs qui ont été capables de trouver une solution avant d'arriver à certain nombre maximal d'itérations (soit 25 000) ou au temps d'expiration (soit 5 minutes).

3. Il est basé sur le code de Daniel Díaz disponible dans https ://sourceforge.net/projects/adaptivesearch/

4. POSL source code is available in https ://github.com/alejandro-reyesamaro/POSL

5. Document supplémentaire *Experiments[02-2016].ods* disponible sur : https ://goo.gl/apsVSF

Instance	$OM_{s'}$ Best Improvement		$OM_{s'}$ First Improvement	
	T	It.	T	It.
5-3-7	4.99	4,421	**1.32**	1322
8-4-7	5.10	954	**1.82**	445
9-4-8	12.37	1,342	**6.43**	873
11-7-5	5.19	351	**2.22**	273

TABLEAU 3 – Expériences avec *Social Golfers* en comparant les fonctions de sélection

Il est important de souligner que POSL ne vise pas à obtenir les meilleurs résultats en termes de performance, mais de donner la possibilité de prototyper rapidement et d'étudier différentes stratégies de recherche coopératives ou non coopératives. Notre objectif n'est donc pas de construire des solveurs plus rapides, mais d'obtenir des moyens plus rapides pour les étudier.

Dans la première étape des expériences, nous utilisons le langage basé sur des opérateurs fournis par POSL pour construire et tester de nombreuses et différentes stratégies non communicatives. L'objectif est de sélectionner la meilleure stratégie pour exécuter des tests avec communication. D'abord, nous nous sommes concentrés sur le choix de la bonne fonction de voisinage. Dans le cas du problème de *Social Golfers*, cette expérience a été lancée en utilisant une stratégie basique montrée dans l'algorithme 4 pour bien nous concentrer sur l'étude des modules et non sur la *computation strategy*. Cette stratégie n'a pas été capable de résoudre les problèmes pour les cas de plus de trois semaines, car elle tombait très souvent dans des minimums locaux. Cela a été la raison pour laquelle nous effectuons cette expérience avec l'instance 10–10–3.

Les résultats du tableau 2 ne sont pas surprenants. L'*operation module* de voisinage OM_V^{AS} est basé sur l'algorithme *Adaptive Search*, qui a montré de très bons résultats. Il sélectionne la variable (joueur) qui contribue le plus au coût et permute sa valeur avec les autres variables (joueurs) pour tous les groupes, et pour chaque semaine. OM_V^{Std} n'utilise aucune information supplémentaire. Il effectue donc chaque permutation possible entre deux joueurs dans les différents groupes, chaque semaine. Cela signifie que ce voisinage est $g \times p$ (nombre de groupes et nombre de joueurs, respectivement) fois plus grand que le précédent. La combinaison avec l'opérateur (ρ) exécute le module OM_V^{Std} ou le OM_V^{AS}, en fonction de la probabilité ρ. La combinaison avec l'opérateur $(\cup)$ est l'union de ces voisinages. Dans ces trois derniers, le temps passé pour la recherche à l'intérieur du voisinage à chaque itération est significatif, même si le nombre d'itérations dans ces trois cas est inférieure à celui qu'utilise le module OM_V^{AS}, car ce dernier effectue de nombreux *Reset* pendant le processus de recherche.

Une fois avoir obtenu la bonne fonction de voisinage, nous nous sommes concentrés sur le choix de la meilleure fonction de *sélection*. Nous avons comparé deux *operation modules* différents en utilisant la *computation strategy* de l'algorithme 7. Cette dernière combine les *operation modules* de sélection ($M_{s'}^{First}$ ou $M_{s'}^{Best}$) avec $M_{s'}^{Rand}$, pour éviter les minimums locaux : elle tente d'améliorer le coût un certain nombre de fois. Si elle n'y arrive pas, elle sélectionne un voisin aléatoire pour l'itération suivante. Le premier *operation module* a été $OM_{S'}^{Best}$ qui sélectionne la meilleure configuration à l'intérieur du voisinage. Il n'a pas seulement passé plus de temps à chercher une meilleure configuration, mais il est également plus sensible pour tomber dans des minimums locaux. Le deuxième *operation module* a été $OM_{S'}^{First}$ qui sélectionne la première configuration à l'intérieur du voisinage en améliorant le coût actuel. En utilisant ce module, la stratégie perd le facteur d'intensification, mais gagne en vitesse et en diversification. Le tableau 3 présente les résultats de cette expérience, en montrant qu'une stratégie orientée plus exploratoire est plus efficace pour le problème de *Social Golfers*.

Dès lors que la meilleure stratégie a été choisie, nous lançons des expériences pour étudier le comportement de POSL pour résoudre les problèmes ciblés dans des scénarios avec communication. Certaines compositions de POSL méta-solveurs ont été prises en compte : i) la structure de la communication (avec–sans communication ou un mélange), et ii) la stratégie de communication codée dans le solveur.

Instance	OP : 1 à 1		OP : 1 à N	
	T	It.	T	It.
5-3-7	1.19	1,156	1.11	1,067
8-4-7	**1.30**	**317**	1.46	347
9-4-8	4.38	**597**	5.51	736
11-7-5	1.76	214	**1.62**	**202**

TABLEAU 4 – Expériences avec *Social Golfers* en testant 100% de communication

Instance	OP : 1 à 1		OP : 1 à N	
	T	It.	T	It.
5-3-7	1.04	1,019	1.04	1,031
8-4-7	1.40	337	1.43	353
9-4-8	4.64	637	5.75	776
11-7-5	1.81	220	1.82	222

TABLEAU 5 – Expériences avec *Social Golfers* en testant 50% de communication

Instance	OP :1 à 1		OP : 1 à N	
	T	It.	T	It.
5-3-7	**0.90**	**881**	1.19	1,170
8-4-7	1.39	341	1.46	352
9-4-8	**4.33**	599	4.53	625
11-7-5	1.99	242	1.63	224

TABLEAU 6 – Expériences avec *Social Golfers* en testant 25% de communication

Chaque fois qu'un POSL méta-solveur est lancé, de nombreux processus de recherche indépendants sont exécutés. Dès qu'une bonne configuration est trouvée, elle est transmise du solveur émetteur au récepteur. À ce moment, si l'information est acceptée, il y a quelques solveurs qui recherchent dans le même sous-ensemble de l'espace de recherche, et la stratégie de recherche devient plus orientée à l'exploitation. Cela peut être problématique si la stratégie conduit souvent les solveurs dans des minimums locaux. Dans ce cas, nous ne perdons pas qu'un seul solveur, mais deux ou plus, en fonction de la stratégie de communication. Nous pouvons éviter ce phénomène par une simple mais efficace opération : si un solveur n'est pas capable de trouver une meilleure configuration à l'intérieur du voisinage avec l'exécution de l'*operation module* $OM_{S'}^{First}$, il en sélectionne une au hasard, en exécutant donc l'*operation module* $OM_{S'}^{Rand}$. En utilisant la communication entre les solveurs, cette stratégie produit un certain gain en terme de temps d'exécution (tableau 3 par rapport aux tableaux 4, 5 et 6 pour *Social Golfers*, et tableau 7 par rapport au tableau 8 pour *Costas Array*). Le pourcentage de solveurs récepteurs qui ont été capables de trouver la solution avant les autres, était important, comme nous pouvons le voir dans les tableaux du document supplémentaire disponible sur Internet [6]. Cela montre que la communication a joué un rôle important lors de la recherche, malgré les frais généraux de la communication entre les processeurs (réception, interprétation des informations, pris des décisions, etc.).

Pour la résolution du problème *Costas Array*, le tableau 8 montre que la stratégie 12 est plus performante que les stratégies 13 et 14. Même si la communication de la stratégie 12 est plus soutenue (à chaque itération), elle est faite au bon moment, après avoir sélectionné la configuration pour la prochaine itération : à cette étape, le solveur recherche une configuration avec un coût global inférieur à la configuration courante. Il décide ainsi s'il doit prendre pour la prochaine itération une configuration du voisinage ou la configuration communiquée. La communication dans les stratégies 13 et 14 est faite au moment de faire le *reset*, donc nettement moins souvent. Cette stratégie empêche aussi de faire correctement le *reset* car à chaque fois il est comparé avec une configuration qui a forcement un meilleur coût. Par contre, le surcoût de la communication est élevé, quand la stratégie 12 est utilisée avec une communication de type 1 à N.

Dans les expériences réalisées, l'information partagée était dans tous les cas la meilleure configuration trouvée. Jusqu'à présent, il n'y a pas de résultat en affirmant que c'est la meilleure stratégie. En fait, [4] montre que la configuration courante n'est pas une information intéressante pour partager entre les solveurs. Voilà pourquoi ce sujet mérite une étude plus approfondie. Nous envisageons dans l'avenir, d'enquêter sur d'autres informations à communiquer, telles que des configurations très coûteuses, afin d'éviter d'autres semblables, directions de recherche à éviter ou à prendre en compte, etc.

6. Document supplémentaire *Experiments[02-2016].ods* disponible sur : https ://goo.gl/apsVSF

STRATÉGIE	T	It.	% réussi
Séquentiel (1 cœur)	2.24	35,299	48.50
Parallèle (20 cœurs)	1.44	24,041	100.00

TABLEAU 7 – Expériences avec *Costas Array* 17 (sans communication)

	100% comm.		OP : 50% comm.	
STRATÉGIE	T	It.	T	It.
Str A : 1 à 1	0.89	10,995	1.12	15,427
Str A : 1 à N	1.04	12,847	1.07	15,086
Str B : 1 à 1	0.85	11,431	0.95	14,441
Str B : 1 à N	0.94	12,723	1.07	15,820
Str C : 1 à 1	1.07	11,948	**0.80**	**10,324**
Str C : 1 à N	1.34	17,473	1.32	19,007

TABLEAU 8 – Expériences avec *Costas Array* 17 avec communication (20 cœurs)

6 Conclusions

Dans cet article, nous avons présenté quelques premiers résultats en utilisant POSL pour résoudre des instances des problèmes *Social Golfers* et *Costas array*. Il a été possible d'implémenter différentes stratégies communicatives et non communicantives, grâce au langage basé sur des opérateurs fournis, pour combiner différents *operation modules*. POSL donne la possibilité de définir des *canaux de communication* reliant dynamiquement des solveurs, étant capable de définir des stratégies différentes en terme de pourcentage de solveurs communicatifs. Les résultats montrent la capacité de POSL à résoudre ces problèmes, en montrant en même temps que la communication peut jouer un rôle décisif dans le processus de recherche.

POSL a déjà une importante bibliothèque d'*operation modules* et d'*open channels* prête à utiliser, sur la base d'une étude approfondie sur les algorithmes méta-heuristiques classiques pour la résolution de problèmes combinatoires. Dans un avenir proche, nous prévoyons de la faire grandir, afin d'augmenter les capacités de POSL.

En même temps, nous prévoyons d'enrichir le langage en proposant de nouveaux opérateurs. Il est nécessaire, par exemple, d'améliorer le langage de *définition du solveur*, pour permettre la construction plus rapide et plus facile des ensembles de nombreux nouveaux solveurs. En plus, nous aimerions élargir le langage des opérateurs de communication, afin de créer des stratégies de communication polyvalentes et plus complexes, utiles pour étudier le comportement des solveurs.

Références

[1] Noga Alon, Chen Avin, M Koucký, Michal Koucky, Gady Kozma, Zvi Lotker, and Mark R. Tuttle. Many Random Walks Are Faster Than One. *Combinatorics, Probability and Computing*, 20(4) :481–502, 2011.

[2] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237 :82–117, jul 2013.

[3] Alexander E.I. Brownlee, Jerry Swan, Ender Özcan, and Andrew J. Parkes. Hyperion 2. A toolkit for {meta- , hyper-} heuristic research. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, pages 1133–1140, Vancouver, BC, 2014. ACM.

[4] Yves Caniou, Philippe Codognet, Florian Richoux, Daniel Diaz, and Salvador Abreu. Large-Scale Parallelism for Constraint-Based Local Search : The Costas Array Case Study. *Constraints*, 20(1) :30–56, 2014.

[5] Daniel Diaz, Florian Richoux, Philippe Codognet, Yves Caniou, and Salvador Abreu. Constraint-Based Local Search for the Costas Array Problem. In *Learning and Intelligent Optimization*, pages 378–383. Springer, 2012.

[6] Alex S Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary computation*, 16(1) :31–61, 2008.

[7] Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, and Christophe Ponsard. Combining Neighborhoods into Local Search Strategies. In *11th MetaHeuristics International Conference*, Agadir, 2015. Springer.

[8] Frédéric Lardeux, Éric Monfroy, Broderick Crawford, and Ricardo Soto. Set Constraint Model and Automated Encoding into SAT : Application to the Social Golfer Problem. 2014.

[9] Simon Martin, Djamila Ouelhadj, Patrick Beullens, Ender Ozcan, Angel A Juan, and Edmund K Burke. A Multi-Agent Based Cooperative Approach To Scheduling and Routing. *European Journal of Operational Research*, 2016.

[10] Danny Munera, Daniel Diaz, Salvador Abreu, and Philippe Codognet. A Parametric Framework for Cooperative Parallel Local Search. In *Evolutionary Computation in Combinatorial Optimisation*, volume 8600 of *LNCS*, pages 13–24. Springer, 2014.

[11] Alejandro Reyes-amaro, Éric Monfroy, and Florian Richoux. POSL : A Parallel-Oriented metaheuristic-based Solver Language. In *Recent developments of metaheuristics*, to appear. Springer.

[12] Alejandro Reyes-Amaro, Éric Monfroy, and Florian Richoux. A Parallel-Oriented Language for Modeling Constraint-Based Solvers. In *Proceedings of the 11th edition of the Metaheuristics International Conference (MIC 2015)*. Springer, 2015.

[13] Alejandro Reyes-Amaro, Éric Monfroy, and Florian Richoux. Un langage orienté parallèle pour modéliser des solveurs de contraintes. In *Onzièmes Journées Francophones de Programmation par Contraintes (JFPC)*, Bordeaux, 2015.

[14] Jerry Swan and Nathan Burles. Templar - a framework for template-method hyper-heuristics. In *Genetic Programming*, volume 9025 of *LNCS*, pages 205–216. Springer International Publishing, 2015.

[15] El-Ghazali Talbi. Combining metaheuristics with mathematical programming, constraint programming and machine learning. *4or*, 11(2) :101–150, 2013.

Annexe A : POSL stratégies

Algorithm 4: Stratégie standard

```
st_std := strategy
  oModules: M_S, M_V, M_SE, M_D;
{ // Iter : number of iterations
  execute(M_S, while(Iter < k1)
                  execute(M_V, execute(M_SE, M_D));
               end
  );
}
```

Algorithm 5: Combinaison de deux fonctions de voisinage en utilisant l'opérateur *RHO*

```
st_rho := strategy
  oModules: M_S, M_V1, M_V2, M_SE, M_D;
{
  execute(M_S, while(Iter < k1)
                  execute( rho(p, M_V1, M_V2),
                           execute(M_SE, M_D)
                  );
               end
  );
}
```

Algorithm 6: Ensemble de deux fonctions de voisinage

```
st_u := strategy
  oModules: M_S, M_V1, M_V2, M_SE, M_D;
{
  execute(M_S, while(Iter < k1)
                  execute( union(M_V1, M_V2),
                           execute(M_SE, M_D)
                  );
               end
  );
}
```

Algorithm 7: Stratégie pour échapper des minimums locaux

```
st_eager := strategy
  oModules: M_S, M_V, M_SE, M_D;
{  // Sci: number of iterations with the same cost
  execute(M_S,
          while(Iter < k1)
            execute(M_V,
                    execute(?(Sci < k2, M_SE, M_SE.Rand),
                            M_D
                    )
             );
           end
  );
}
```

Algorithm 8: Stratégie communicative (émetteur)

```
st_eager_sender := strategy
  oModules: M_S, M_V, M_SE, M_D;
{
  execute(M_S,
          while(Iter < k1)
            execute(M_V,
                    execute(?( Sci < k2,
                               send(M_SE),
                               M_SE.Rand),
                            M_D
                    )
            );
          end
  );
}
```

Algorithm 9: Stratégie communicative (récepteur)

```
st_eager_sender := strategy
  oModules: M_S, M_V, M_SE, M_D;
  oChanels: Ch_DP;
{
  execute(M_S,
          while(Iter < k1)
            execute(M_V,
                    execute(?( Sci < k2,
                               ?( Sci < k3,
                                  min(M_SE, Ch_DP) ,
                                  M_SE),
                               M_SE.Rand),
                            M_D
                    )
            );
          end
  );
}
```

Algorithm 10: Stratégie basée sur des Resets

```
st_hard := strategy
  oModules: M_S, M_R, M_V, M_SE, M_D;
{
  execute(M_S,
          while(Iter < k1)
            execute(M_R,
                    while(Iter < k2)
                      execute(M_V, execute(M_SE, M_D));
                    end
            );
          end
  );
}
```

Algorithm 11: Stratégie basée sur des Resets (émetteur)

```
 st_hard_sender := strategy
oModules: M_S, M_R, M_V, M_SE, M_D;
{
  execute(M_S,
          while(Iter < k1)
            execute(M_R,
                    while(Iter < k2)
                      execute(M_V, execute(M_SE, send(M_D)));
                    end
            );
          end
  );
}
```

Algorithm 12: Stratégie A basée sur des Resets (récepteur)

```
st_hard_receiver_a := strategy
  oModules: M_S, M_R, M_V, M_SE, M_D;
  oChannels: Ch_Sol;
{
  execute(M_S,
          while(Iter < k1)
            execute(M_R,
                    while(Iter < k2)
                      execute(M_V, execute(M_SE,
                                           min(M_D, Ch_Sol)
                                  )
                      );
                    end
            );
          end
  );
}
```

Algorithm 13: Stratégie B basée sur des Resets (récepteur)

```
st_hard_receiver_b := strategy
  oModules: M_S, M_R, M_V, M_SE, M_D;
  oChannels: Ch_Sol;
{
  execute(M_S,
          while(Iter < k1)
            execute(?(Iter % k2, M_R, min(M_R, Ch_Sol)),
                    while(Iter < k2)
                      execute(M_V, execute(M_SE, M_D)));
                    end
             );
          end
  );
}
```

Algorithm 14: Stratégie C basée sur des Resets (récepteur)

```
st_hard_receiver_c := strategy
  oModules: M_S, M_R, M_V, M_SE, M_D;
  oChannels: Ch_Sol;
{
  execute(M_S,
          while(Iter < k1)
            execute(min(M_R, Ch_Sol),
                    while(Iter < k2)
                      execute(M_V, execute(M_SE, M_D));
                    end
             );
          end
  );
}
```

La classification des gènes basée sur les CSP pondérés

Ramzi Ben Mhenni[1] **Amel Mhamdi**[2] **Wady Naanaa** [1]

[1] Faculté des sciences de Monastir, Tunisie
[2] Faculté des sciences économiques et de gestion de Sfax, Tunisie
benmhenniramzi@gmail.com amel.mhamdi2@gmail.com wady.naanaa@fsm.rnu.tn

Résumé

La classification des objets par rapport à une similitude ou une distance donnée est un problème souvent rencontré dans des problèmes de bio-informatique. Plusieurs algorithmes de classification bien connus, tel que les algorithmes de partitionnement des gènes similaires, sont basés sur la transformation de la matrice d'entrée vers un graphe pondéré bien que le problème d'édition de graphe pondéré résultant est NP-difficile. Le but est de transformer le graphe en entrée en une union de cliques disjointes telles que la somme des poids de toutes les arêtes modifiées soit minimisée.

Dans cet article, Nous présentons une nouvelle approche pour le problème de classification des gènes en terme de problème de satisfaction de contrainte pondéré (WCSP).

Une étude comparative avec les deux approches les plus utilisées pour la résolution du problème de classification des gènes similaires, qui sont les algorithmes à paramètre fixe et les algorithmes s'appuyant sur la programmation linéaire en nombre entier, sera proposée. Nous montrons que l'approche proposée permet souvent de trouver une solution optimale en temps raisonnable. Nous avons appliqué notre algorithme à des données biologiques.

Abstract

The classification of objects relative to a similarity or a given distance is a central problem in computational biology. Several well-known classification algorithms, such as partitioning problem of similar genes, are based on the transformation of the input matrix into a weighted graph, although the resulting weighted cluster editing problem is NP-hard. The goal is to transform the input graph into a union of disjoint cliques such as the sum of the weights of all modified edges is minimized.

In this article, we propose an encoding of the classification of similar genes in terms of a Weighted Constraint Satisfaction Problem (WCSP), which we solve by an adapted version of an efficient constraint solver. We compare both quality and running times of these algorithms on protein similarity graphs. A comparison of our method with an exact fixed-parameter tractability method and an integer linear programming algorithm, two of the most used algorithms, shows that our approach is able to solve large instances with several thousand edge modifications.

1 Introduction

Le problème de similarité des gènes, est défini comme suit : Deux gènes sont considérés comme identiques, s'ils dérivent par descendance mendélienne, en l'absence de mutation, d'un même gène ancêtre. L'identité de gènes est une relation d'équivalence qui induit sur tout ensemble de gènes une partition en sous ensembles disjoints appelés classes d'identité. Les gènes appartenant à une même classe d'identité sont considérés comme identiques. L'avantage de la partition en classe d'identité est que la connaissance de la nature d'un seul gène de la classe permet de déterminer la nature de tous les autres de la même classe, car il existe entre eux une relation de parenté : ils sont la copie non modifiée par mutation d'un même modèle, le gène ancêtre. Deux gènes appartenant à deux classes d'identité distinctes sont indépendants, car la connaissance de la nature de l'un ne permet pas de déterminer avec certitude la nature de l'autre [6]. La formulation des données de ce problème par un graphe est assez évidente. Ainsi, chaque sommet représente un gène et chaque arête reflète la similarité entre deux gènes. Cette formulation fait intervenir des familles de graphes très connues qui sont les graphes d'intervalles, les graphes de comparabilité, les graphes triangulés ou faiblement triangulés, etc)

[5]. Il est alors souvent possible d'en déduire des algorithmes polynomiaux, répondant au problème posé, en s'appuyant sur la littérature très fournie dans le domaine de la théorie des graphes.
Dans ce travail, nous nous intéressons aux graphes issue des données de similarité des protéines de la base de données COG [3]. Ces données peuvent être codées par les graphes, où les séquences de protéines constituent les sommets. Une arête relie un sommet A à un sommet B si les deux séquences A et B ont une valeur de similarité au-dessus d'un seuil fixe. Ceci donne lieu à un problème bien connu qui est celui de l'édition de graphe pondéré "weighted cluster editing" (WCE) [14] [7]. Les graphes ainsi obtenus sont non orientés pondérés tels que le coût sur les arêtes représente la mesure de similarité entre les protéines. Une arête entre deux sommets A et B signifie que A et B sont similaires. Pour déterminer une partition des sommets, une fonction de coût est utilisée pour spécifier le coût de modification sur les arêtes (insertion/suppression). Le but est de trouver les modifications d'arête qui engendrent un coût total minimum telles que le graphe cible est une union de cliques disjointes. Ceci suppose qu'une classification correcte correspond à un graphe de clique, ou "cluster graph", c'est-à-dire, un graphe qui contient seulement des cliques disjointes. On peut donc supposer que le graphe source contient certaines arêtes inutiles qui sont assimilées à du "bruit". Lorsque nous cherchons à partitionner un graphe en cliques disjointes et lorsqu'il y a peu de sommets à partitionner, le nombre de combinaisons à explorer est faible et le problème peut être résolu rapidement. Cependant, l'ajout de quelques sommets supplémentaires au graphe peut augmenter considérablement le nombre de combinaisons, de sorte que le temps de résolution devient excessivement long.

Le problème d'édition de graphe pondéré est également connu comme le problème de projection de graphe transitif et a été résolu pour des graphes particuliers par Zahn [16]. En 1986, Krivanek et Moravek ont prouvé la NP-complétude du problème de la classification d'arbre hiérarchique, dont l'édition de graphe est un cas particulier [1]. Plusieurs années plus tard, Shamir et al. ont montré la NP-complétude du problème d'édition de graphe pondéré [20]. Plusieurs algorithme heuristiques ont étés conçus pour la résolution de ce problème tel que HCS [19], CLICK [21], CAST [18], et FORCE [22]. CAST essaie de trouver la solution optimale avec une grande probabilité. A leurs tours, HCS et CLICK s'appuient sur des coupes minimales pour trouver une solution approchée alors que FORCE utilise un algorithme de traçage du graphe (graph layout algorithm).

Récemment, la complexité paramétrée de ce problème, utilisant le nombre d'arêtes modifiées comme paramètre, a gagné beaucoup d'attention [9] [10] [11] [12][13]. Les algorithmes dédiés à des problèmes classés comme problèmes traitables à paramètre fixe (FPT), utilisent le coût désiré comme paramètre pour contrôler la complexité temporelle. En effet, le paramètre de contrôle est étroitement lié à la complexité temporelle des algorithmes FPT. Cette complexité est, en général, de la forme $f(k)n^{O(1)}$, où k est un paramètre spécifique du problème, n est la taille de l'instance et $f(k)$ est une fonction de k. Rahmann et al. [8] ont proposé un algorithme à paramètre fixe pour la résolution du WCE en utilisant les coûts de modification comme paramètre avec un temps d'exécution de $O(3^k + |V|^3 \log |V|)$. Dans [7], les auteurs donnent une stratégie de branchement raffinée avec un temps d'exécution de $O(2.42^k + |V|^3 log|V|)$. Dans [14], l'auteur introduit une nouvelle stratégie qui conduit à l'algorithme le plus efficace, pour le problème de WCE, en s'appuyant sur les règles, dites, de réduction présentées dans [7].
L'algorithme FPT parcourt un arbre de recherche avec un nombre fixe des modifications autorisées k. A chaque changement, on introduit ou on supprime des arêtes de l'instance toute en décrémentant le paramètre k. L'algorithme s'arrête lorsque le paramètre k atteint zéro ou lorsque l'instance courante ne contient pas le sous-graphe P_3 (c'est-à-dire trois sommets liées par deux arêtes). Si l'algorithme réussit à trouver une solution en introduisant moins de k modifications alors le but est atteint. Dans le cas contraire, le paramètre fixe est incrémenté et une nouvelle passe de l'algorithme FPT est exécutée. Le présent travail est motivé par celui présenté dans [14]. Ce dernier a introduit deux contributions : un algorithme à paramètre fixe et une approche se basant sur la programmation linéaire. Les auteurs ont comparé les performances de l'algorithme FPT et l'algorithme ILP basé sur la stratégie de "branch and cut". Leurs résultats indiquent que les deux approches sont susceptibles de résoudre de grands graphes avec des milliers de modifications d'arêtes.
Dans le cas du problème de similarité des protéines ou du problème d'édition de graphe pondéré, la plupart des approches existantes partent d'un graphe simple et cherchent à partitionner ce dernier en un ensemble des cliques disjointes avec le minimum de nombres de suppressions des arêtes k.
Par ailleurs, de nombreux problèmes d'optimisation combinatoire telles que le problème de similarité des protéines ou le problème d'édition de graphe pondéré peuvent être formulés comme des Problèmes de

Satisfaction de Contraintes Valuées (VCSPs) [23], où les contraintes sont définies à l'aide de fonctions coût reflétant différents degrés de satisfaction. Résoudre un VCSP revient à trouver une attribution de valeurs aux variables ayant un coût global optimal. Cette tâche est difficile du point de vue de la complexité algorithmique puisqu'il s'agit d'un problème NP-difficile.

Dans ce travail, le but est de proposer une nouvelle approche qui s'appuie sur le formalisme des problèmes de satisfaction de contraintes valuées pour résoudre le problème de similarité des séquences protéiques. Ce formalisme offre un cadre rigoureux, plus général que celui des CSP et surtout des algorithmes performants pour résoudre efficacement les problèmes de classification faisant intervenir des fonctions coût.
L'algorithme de classification proposé sera appliqué sur des instances réelles qui codent des données biologiques issues du problème de similarité des séquences protéiques issues du problèmes de satisfaction de contraintes pondéré (WCSP). Les instances obtenues sont et par la suite, résolues au moyen d'un algorithme de branch-and-bound couplé avec une stratégie heuristique d'élargissement itérative (IB). La résolution est effectuée à l'aide du solveur de contraintes Toulbar2.

Le reste de l'article est organisé comme suit. Dans la Section 2 nous introduisons les notions de base ainsi qu'une présentation synthétique des concepts liés aux problème de similarité des séquences protéiques le principal problème qui peut se ramener à une partition des protéines similaire. Pour ce qui est des outils utilisés, nous décrivons le problème d'édition de graphe pondéré, le problème de satisfaction des contraintes pondéré, ainsi que les notations générales nécessaires à la bonne compréhension du reste du document. La Section 3, détaillera la modélisation du problème de similarité des séquences protéiques en terme de WCSP. La Section 4 décrit l'algorithme de solution proposée. Dans la Section 5, une comparaison expérimentale de notre approche et celle des approches FPT et ILP pour le problème de similarité des séquences protéiques sera présentée. Finalement, nous concluons en présentant un bilan du travail et en discutant des perspectives.

2 Définitions et notations

2.1 Regroupement des données biologiques

L'ADN (deoxyribonucleic acid) est le support de l'information génétique. C'est le "plan détaillé" de notre organisme aussi appelé code génétique. Il contient toutes les informations nécessaires au développement et au fonctionnement du corps.

Un gène est un morceau de cet ADN qui correspond à une information génétique particulière. Les gènes portés par l'ADN sont codés sous une autre forme au cours d'un processus nommé "transcription". Lors de la transcription du gène, un des brins d'ADN est transcrit en séquence ARN. Cette copie, appelée ARN messager (ARNm), est destinée à l'usine de fabrication des protéines et lui fournit la recette de la protéine codée par le gène. Les protéines fonctionnelles sont le plus souvent synthétisées à partir des gènes par traduction directe d'un ARN messager. Cependant, lorsqu'une protéine doit être produite très rapidement ou en grande quantité, c'est tout d'abord un précurseur protéique qui est produit par l'expression du gène. La notion de profil d'expression peut ainsi par extension désigner l'ensemble des gènes qui partagent un même profil.

Les niveaux d'expression mesurés pour chaque gène dans un certain nombre de situations constituent ce que l'on appelle classification du profil d'expression de ce gène pour un jeu de situations donné. Des méthodes d'analyse standard permettent de regrouper par classification hiérarchique des gènes d'une même expérience qui présentant des profils d'expression similaires.

Plusieurs recherches sont motivés par l'idée que les gènes ayant des profils d'expression similaires participent à un même processus biologique. Le but est de regrouper les gènes impliqués dans un même processus biologique. c'est-à-dire l'ensemble de gènes ayant des profils d'expression similaires. les profils d'expression de deux gènes similaires peuvent être expliquées par une fonction similaire de l'ARN ou de la protéine correspondante. Cela explique pourquoi le regroupement des gènes similaires en fonction des profils d'expression est essentielle pour l'analyse moderne des données biologiques.

Plusieurs approches de classification ont été publiées pour traiter ce problème [19] [21] [18] [22].

En outre, le regroupement de données est une tâche classique en biologie. Son objectif est de partitionner un ensemble de données en clique de telle sorte que les éléments d'une même clique sont plus similaire selon un ou plusieurs critère que les éléments de différentes autres cliques. Une stratégie commune pour la classification est de choisir un seuil de similarité et construire un graphe correspondant selon les règles suivantes : les protéines se réfèrent aux sommets du graphe, et une arête relie deux sommets si et seulement si la similitude entre les deux sommets est supérieur à un seuil donné. La Figure 1 illustre ce processus.

Pour deux protéines A et B une similitude dépassant un certain seuil implique que A et B ont un ancêtre commun. Nous relions les sommets A et B par une arête "similaire" et nous écrivons $A \sim B$. Cette

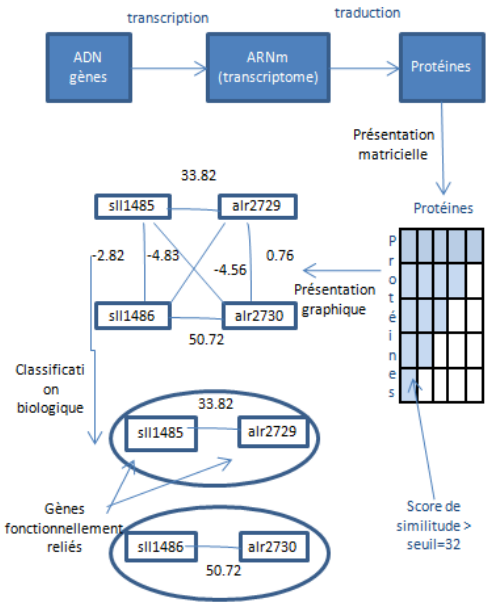


FIGURE 1 – Illustration du processus de regroupement biologique.

propriété est transitive, elle peut faire appel à des séquences intermédiaires. Cela signifie qu'on peut déduire la similarité entre les protéines A et C de l'existence d'une troisième protéine B, de telle sorte que A et B, ainsi que B et C sont similaires, si la similitude entre A et B, ainsi que celle entre B et C est supérieure au seuil mentionné. Cependant, le graphe obtenu n'est pas nécessairement transitif, c'est-à-dire que $A \sim B$ et $B \sim C$ n'implique pas nécessairement $A \sim C$. Notre objectif est de convertir le graphe construit en un graphe composé seulement par des cliques disjointes, un graphe transitif donc, avec le minimum de coûts (minimum d'insertion / suppression) d'arêtes.

Nous préconisons que le problème d'édition de graphe est bien adapté au problème de classification de protéines tel que les sommets adjacents représentent des gènes similaires.

2.2 Présentation du formalisme VCSP

Un problème de satisfaction de contraintes se défini par la donnée d'un ensemble de variables, ayant chacune un domaine discret de valeurs de taille finie, puis par un ensemble de contraintes qui mettent en relation les variables et définissent des combinaisons de valeurs compatibles. Un Problème de Satisfaction de Contraintes Valuées est alors une extension du Problème de Satisfaction de Contraintes classique en associant des coûts aux différentes combinaisons de valeurs que peuvent prendre les variables. Ces coûts sont appelés valuations [23].
Pour définir un VCSP, en plus des composantes définissant un CSP, on utilise une structure de valuation S. Elle associe une valuation à chaque combinaison de valeurs que peuvent prendre les variables.

Définition 1 *Une structure de valuation est un triplet $S = (E, \oplus, \preceq)$ où :*
- *E est l'ensemble des valuations contenant un élément minimal $\bot$ et un élément maximal $\top$;*
- *$\preceq$ est un ordre total sur E ;*
- *$\oplus$ est un opérateur commutatif, associatif et monotone.*

Le Problème de Satisfaction de Contraintes Valuées (VCSP) est alors défini comme suit :

Définition 2 *Un Problème de Satisfaction de contraintes Valuées (VCSP) est défini par (X, D, C, S) où :*
- *X est un ensemble fini de variables ;*
- *D est un ensemble fini de domaines, $D_x \in D$ étant le domaine de $x \in X$;*
- *C est un ensemble de contrainte valuées. Chaque contraintes est un couple (σ, ϕ) où $\sigma \subseteq X$ est la portée de la contrainte et $\phi : \Pi_{x \in \sigma} D_x \rightarrow E$ est une fonction de valuation ;*
- *$S = (E, \oplus, \preceq)$ est une structure de valuation.*

Une variable $x \in X$ doit être affectée à une seule valeur de son domaine, D_x. Si une contrainte valuée est définie par une fonction dont le domaine est un sous-ensemble de $\{\bot, \top\}$, alors il s'agit d'une contrainte dure, sinon, il s'agit d'une contrainte souple. L'arité d'une contrainte valuée est la taille de sa portée. L'arité d'un problème est l'arité maximal de toutes ses contraintes.
La valuation d'une affectation t à un sous ensemble de variables $V \subseteq X$ est donné par

$$\Phi_{\mathcal{P}}(t) = \bigoplus_{(\sigma,\phi) \in C, \sigma \subseteq V} \phi(t \downarrow \sigma) \qquad (1)$$

où $t \downarrow \sigma$ désigne la projection de t sur les variables de σ. Par conséquent, une solution optimale globale pour un VCSP sur n variables est un n-tuplet t tel que $\Phi_{\mathcal{P}}(t)$ est minimal sur tous les n tuplets possibles.

Un VCSP a de nombreuses variantes qui diffèrent principalement par la structure de valuation utilisée. Le choix de la structure de valuation la plus appropriée dépend des caractéristiques du problème à formuler en termes de VCSP. Pour formuler le problème WCE,

nous avons opté pour la variante dite WCSP (pour Weighted Constraint Satisfaction Problem), qui vise à déterminer une solution qui satisfasse le plus grand nombre de contraintes possible.

2.3 Édition de graphe pondéré

Le concept de regrouper des données a longuement été étudié dans le cadre de nombreux contextes et disciplines. Quand les données sont sous forme de graphe, le regroupement de ces données peut se traduire par un partitionnement du graphe associé.
L'ensemble des sommets V du graphe source correspondent aux objets à classifier. Une arête entre deux sommets suppose une certaine similarité entre ces deux objets. Depuis, plusieurs variantes problèmes d'édition de graphes ont vu le jour, ce qui a permis à différents algorithmes d'approximations et heuristiques d'être formulés. Le problème d'édition de graphes est utilisé dans différents domaines comme la biologie, le traitement d'image, les bases de données et surtout la bio-informatique.

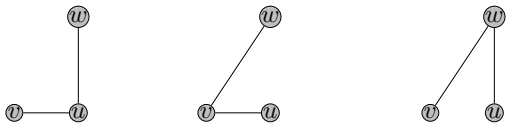


FIGURE 2 – Différentes cas d'un P_3.

Dans ce travail, nous focalisons sur les graphes non orientés, sans boucles et sans arêtes multiples mais où les arêtes sont pondérés. Le problème d'édition de graphe pondéré peut alors être défini comme suit :

Définition 3 *Étant donné un graphe non orienté, pondéré $G = (V, c)$ où $c : \mathcal{P}_2(V) \to \mathbb{R}$ est une fonction qui retourne les coûts de modification pour chaque couple $\{u, v\}$ de $\mathcal{P}_2(V)$, trouver un ensemble de modifications (insertions et suppression) avec le minimum de coût total, de sorte que le graphe modifié soit un graphe de cluster.*

Nous désignons par *graphe de clusters* un graphe composé d'un ensemble de cliques disjoint. Un graphe de clusters ne peut contenir aucun P_3 induit, où P_3 est un graphe composé de trois sommets u, v, et w tel que, $\{u, v\}$ et $\{u, w\}$, sont deux arêtes de G mais pas, $\{v, w\}$ (voir Figure 2). Dans ce qui suit, nous utilisons une version légèrement modifiée s de la fonction coût c. Si $\{u, v\} \in E$ alors, $s(u, v) := c(u, v)$, de sorte que $s(u, v) \geq 0$. Si $\{u, v\}$ est absent de E, alors $s(u, v) := -c(u, v)$, de sorte que $s(u, v) \leq 0$.

Nous définissons, à présent, la version paramétrée du problème d'édition de graphe pondéré avec le coût total de modification comme paramètre k :

Définition 4 *Étant donné un graphe non orienté, pondéré $G = (V, c)$ où $c : \mathcal{P}_2(V) \to \mathbb{R}$ et un entier k, trouver un ensemble de modifications d'arêtes (insertions et suppressions) de sorte que le coût total soit inférieur ou égal à k, et tel que le graphe modifié soit un graphe de clusters.*

Le problème d'édition de graphes pondérés consiste à déterminer, pour un graphe G et paramètre k, la possibilité d'effectuer, au plus, k modification dans G afin d'obtenir une union de cliques disjointes.

3 L'édition de graphe pondéré en tant que WCSP

Dans le but d'avoir un problème de taille raisonnable, nous proposons un codage pour le problème d'édition de graphe pondéré en terme de problème de satisfaction de contrainte pondéré (WCSP). Les instances obtenues seront, résolues par un solveur de contraintes Toulbar2 [17].

Nous devons, tout d'abord, identifier l'ensemble des variables X, les domaines de valeurs des variables D, et enfin l'ensemble des contraintes C. La formulation proposée, WCSP-WCE, associe une variable à chaque sommet dans G. Des contraintes binaires souples seront utilisées afin de sanctionner la formation de sous-graphe P_3 et de limiter les modifications portées sur le graphe d'origine. Un deuxième objectif consiste à limiter le nombre de cliques créés dans le graphe cible. Cet objectif sera atteint grâce à l'utilisation de la stratégie d'élargissement itératif (IB) [15].

3.1 Les variables et leurs domaines

Considérons un graphe pondéré non orienté définit par $G = (V, E)$. Nous avons donc $|X| = |V|$. Le domaine de chaque variable $x \in X$ est l'ensemble de toutes les cliques possibles à utiliser. Nous avons donc $D_x = \{1, ..., cmax\}$, pour tout $x \in X$. Un algorithme glouton [2] nous a permit de calculer le nombre maximal de cliques dans le graphe cible ($cmax$).

3.2 les contraintes

Le seul type de contrainte qui sera considéré dans cette partie est la contrainte binaire.
Afin d'interdire la formation de sous-graphe P_3 et de minimiser le nombre de modification d'arêtes, nous utilisons des contraintes binaires. Interdire le sous-graphe P_3 peut être atteint par l'application de la règle suivante : si $\{v_i, v_j\} \in E$ alors v_i et v_j doivent de préférence être placé dans la même clique dans le graphe cible. Cette règle peut être forcée en créant

TABLE 1 – La fonction d'évaluation de contraintes binaire.

	$x_i = x_j$	$x_i \neq x_j$
$\{v_i, v_j\} \in E$	0	$s(v_i v_j)$
$\{v_i, v_j\} \notin E$	$s(v_i v_j)$	0

une contrainte binaire souple pour chaque paire de variable du WCSP-WCE. La fonction de valuation de ces contraintes binaires est donnée dans la Table 1. Nous distinguons deux situations qui impliquent des coûts : (1) le cas où $\{v_i, v_j\} \in E$ et la décision est de supprimer l'arête $\{v_i, v_j\}$ du graphe d'origine en mettant v_i et v_j dans deux cliques différents. (2) le cas où $\{v_i, v_j\} \notin E$ et la décision est de mettre v_i et v_j dans le même clique.
Soient x_i et x_j deux variables du WCSP-WCE associées respectivement aux sommets v_i et v_j. Nous devons avoir, dans le codage WCSP-WCE, une contrainte binaire dont la portée est la paire $\{x_i, x_j\}$. Cette contrainte est soit une égalité ou une contrainte souple d'inégalité est définie selon que $\{v_i, v_j\}$ est dans E ou non. Ainsi la fonction de valuation des contraintes d'égalité souple est définie par :

$$\phi_{eq}(a,b) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{si } a = b \\ s(v_i v_j) & \text{sinon} \end{cases} \quad (2)$$

Et la fonction de valuation des contraintes d'inégalité souple est définie par :

$$\phi_{neq}(a,b) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{si } a \neq b \\ s(v_i v_j) & \text{sinon} \end{cases} \quad (3)$$

3.3 Exemple

Un exemple est décrit dans la Figure 3, où l'entrée est un graphe de 5 sommets et 6 arêtes. Le WCSP-WCE corresponde à cinq variables. Selon la méthode de recherche gloutonne (Greedy method search) décrite dans [2], le regroupement peut être obtenu en utilisant, au plus, deux cliques qui est le nombre optimal. Le WCSP-WCE a donné lieu à dix contraintes binaires : six contraintes d'égalité souples et quatre contraintes d'inégalités souples. Le coût de la solution donnée pour l'édition de graphe par le modèle de programmation par contrainte est 67.7 puisque il y a deux arêtes du coût 32.8 et 34.9 supprimés. Nous distinguons deux cliques disjointes.

4 Algorithme de résolution

En utilisant le modèle présenté à la Section 3, nous pouvons maintenant résoudre le problème de partitionnement biologique de protéines par l'algorithme le plus utilisé pour la résolution des VCSPs : l'algorithme qui maintient l'arc-consistance directionnelle existentielle MEDAC* [24].

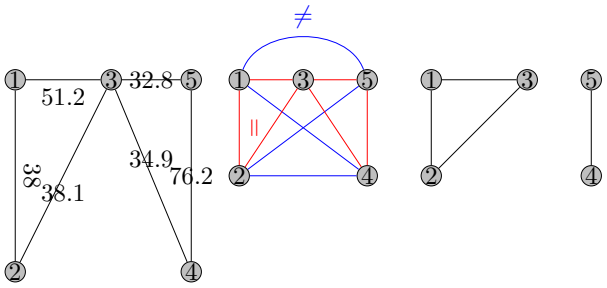


FIGURE 3 – A Gauche : Une instance d'édition de graphe pondéré. Au Centre : Le WCSP correspondant. Les arêtes rouges présentent une contrainte souple d'égalité et les arêtes bleues présentent une contrainte souple d'inégalité. A Droite : une solution optimale.

Les problèmes de satisfaction de contraintes pondéré sont souvent résolu par un algorithme de branch-and-bound. Pour obtenir un algorithme de solution plus efficace, nous avons utilisé l'algorithme de branch-and-bound avec la stratégie d'heuristique d'élargissement itératif (IB)[15].

L'algorithme proposé WCSP-WCE effectue une recherche itérative tout en appliquant la technique de branche-and-bound à chaque nœud de l'arbre de recherche. Un schéma itératif d'élargissement est effectué en exécutant à chaque itération l'algorithme du maintient d'arc-consistance directionnelle existentielle MEDAC* jusqu'à ce qu'une solution optimale est obtenue.

proposition 1 *Soit I une instance de la WCE pour la quel la meilleure solution qui n'utilise plus que k cliques a un coût c. Si I n'a pas de solution partielle qui utilise $k+1$ cliques et dont le coût est inférieur à c, alors c est optimal.*

Preuve.
Supposons que I n'a pas de solution partielle qui utilise $k+1$ cliques, dont le coût est inférieur à c, mais au même temps, c n'est pas le coût optimal. Cela implique qu'il doit y existe une solution $S^{'}$ dont le coût c', est inférieur à c.

En outre, S' doit utiliser plus que k cliques, puisque la solution optimale qui n'utilise pas plus que k cliques a coûté c.

Considérons alors que toute solution partielle S_p extrait de $S^{'}$ de telle sorte que S'_p utilise exactement $k+1$ cliques. Observer que le coût de $c^{'}_p$ ne peut pas dépasser

$c^{'}$, et il est moins que c. Cela contredit de l'hypothèse de départ.

La proposition ci-dessus permet d'arrêter la recherche itératif, si elle ne permet pas, d'améliorer la solution courante à une itération donnée.

5 Résultats expérimentaux

Dans cette section, nous évaluons la performance du modèle WCSP et son algorithme associé WCSP-WCE, qui est implémenté en utilisant Toulbar2 [17]. Pour FPT-WCE nous avons utilisé l'outil "software peace" disponible à [3] et pour ILP-WCE nous avons utilisé l'outil "Yoshiko" disponible à [4].

Le WCSP-WCE est comparée avec l'algorithme FPT-WCE et ILP-WCE décrit dans [14]. Les instances biologiques résultants sont transformés en WCSP en appliquant le codage décrit à la section précédente. La figure 4 illustre ce processus.
La Table 2 fournit les résultats obtenus par FPT-WCE, ILP-WCE et WCSP-WCE. Une heure de temps d'exécution a été consacré à chaque instance. Pour certaines instances, aucun des trois algorithmes ne surpasse systématiquement les autres. WCSP-WCE domine légèrement les deux autres algorithmes (FPT-WCE et ILP-WCE) sur les instances peu large ($42 < n < 70$). Les autres instances, ($n < 42$), sont très faciles à résoudre et aucun algorithme ne domine l'autre. WCSP-WCE est toujours plus rapide que ILP-WCE. En effet, sur l'instance où $n = 180$ seul ILP-WCE n'a pas réussi de trouver une solution. Sur des autres instances ($n = 107, n = 108, n = 122$ et $n = 130$), qui sont plus difficiles à résoudre (de taille large $n > 100$) FPT-WCE a trouver une solution dans un temps plus raisonnable que ILP-WCE et WCSP-WCE. Ce résultat peut être expliqué par l'importance des règles de réduction qui s'exécute en temps polynomial. Pour le reste des solutions, WCSP-WCE surpasse les deux autres algorithmes.

paramétres		FPT-CE		ILP-CE		WCSP-CE	
n	moy.$\|E\|$	Coût	Tp	Coût	Tp	Coût	Tp
4-10	4-39	7.561	0.001	7.561	0.466	7.561	0.01
11-18	19-143	29.361	0.001	29.361	0.102	29.361	0.01
21-30	113-359	123.710	0.015	129.010	14.127	129.010	0.11
31-40	62-658	162.649	0.020	162.649	0.128	162.649	0.001
42-50	244-1218	339.488	0.064	339.488	0.302	339.488	0.17
52-60	204-1593	497.383	0.078	497.383	0.430	497.383	0.01
61-70	358-2170	631.255	0.122	631.255	2.082	631.255	0.97
71-79	879-3078	962.571	0.276	962.571	4.357	962.571	1.28
83-97	704-2897	1167.467	0.354	1167.467	1.416	1167.467	1.77

Table 2 – Résultats sur des instances biologiques, le temps CPU en secondes et le coût de la meilleure solution trouvée sont rapportés. Le temps d'exécution a été limité à une heure par instance. Un tiret indique que le programme n'a pas terminé au bout d'une heure de temps CPU.

paramétres		FPT-CE		ILP-CE		WCSP-CE	
n	moy.$\|E\|$	Coût	Tp	Coût	Tp	Coût	Tp
107	857	398.176	0.20923	398.176	0.476	398.176	0.28
108	2809	3726.93	1.68854	3726.93	14.449	3726.93	2.19
113	3087	3734.29	0.488789	3734.29	0.593	3734.29	0.158
122	1573	2202.85	1.51195	2154.75	139.206s	2154.75	242.13
124	3521	536.706	1.13786	536.706	1.448	536.706	1.113
130	2436	1357.97	0.975343	1357.97	0.898	1357.97	1.18
134	7062	3816.23	2.17035	3816.23	6.892	3816.23	2.11
154	5087	2081.98	2.21233	2081.98	8.98	2081.98	2.18
169	12745	2061.93	3.02989	2061.93	1.55	2061.93	2.8
179	6847	6655.19	3.15734	6655.19	1.799	6655.19	2.71
180	3059	6717.59	3.36683	-	-	6717.59	3.15
245	20178	12311.3	8.02822	12311.3	576.682s	12311.3	7.48

Table 3 – Résultats sur des instances biologiques avec $n > 100$, le temps CPU en secondes et le coût de la meilleure solution trouvée sont rapportés. Le temps d'exécution a été limité à une heure par instance. Un tiret indique que le programme n'a pas terminé au bout d'une heure de temps CPU

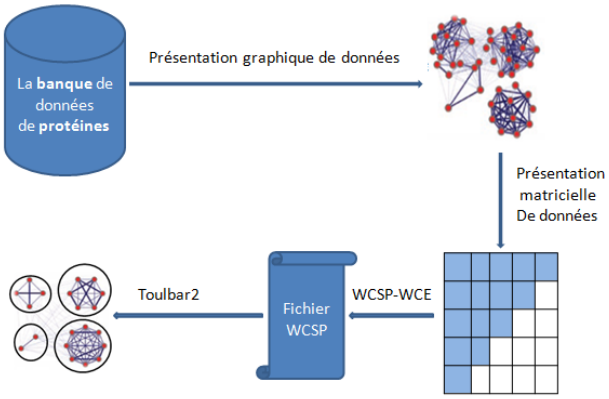


FIGURE 4 – Illustration des étapes de partitionnement de protéines en des cliques disjointes par le codage WCSP-WCE.

6 Conclusion

Dans cet article, nous avons proposé une modélisation en terme de WCSP binaire pour le problème de classification des gènes similaires et sa généralisation en théorie des graphes qui est celui d'édition de graphe pondéré "weighted cluster editing" (WCE). Ensuite, nous avons exploité l'algorithme de filtrage EDAC* intégré à un algorithme de branch and bound pour avoir une solution optimale. Nous avons adopté une stratégie heuristique de l'élargissement itératif es domaines des variables. Enfin, nous avons montré expérimentalement que WCSP-WCE peut être efficace sur des instances de taille moyenne et large.
Une extension possible de ce travail consiste à déterminer les familles de graphes pour lesquels le problème WCE peut être résolu en effectuant uniquement un filtrage (EDAC*).

Références

[1] M. Krivanek and J. Moravek. NP-hard problems in hierarchical-tree clustering, In *Acta Informatica*, 1986

[2] D. Frank, A. Langston Michael, L. Xuemei, P. Sylvain, S. Peter, Z. Yun. The Cluster Editing Problem : Implementations and Experiments. In *IWPEC*, 2006.

[3] https ://bio.informatik.uni-jena.de/software/peace/

[4] http ://www.mi.fu-berlin.de/w/LiSA/YoshikoCharles

[5] Y. Gao, D. R. Hare, and J. Nastos. The cluster deletion problem for cographs In *Discrete Mathematics*, 2013

[6] M. Gillois. La relation d'identité en génétique In *Université de Paris*, 1964

[7] S. Bocker, S. Briesemeister, Q. B. A. Bui, and A. TruB. A fixed-parameter approach for Weighted Cluster Editing In *Asia-Pacific Bioinformatics Conference, APBC 2008*, Kyoto, Japan, 2008

[8] J. Gramm, J. Guo, F. Huffner, and R. Niedermeier. Automated Generation of Search Tree Algorithms for Hard Graph Modification Problems In*Algorithmica*, 2004)

[9] P.Damaschke. Fixed-Parameter Enumerability of Cluster Editing and Related Problems In*Theory Comput. Syst.* 2004

[10] J. Gramm, J. Guo, J., F. Huffner and R. Niedermeier. Graph-Modeled Data Clustering : Fixed-Parameter Algorithms for Clique Generation In*ciac*, 2006)

[11] J.Chen and J. Meng. A 2k kernel for the cluster editing problem In*J. Comput. Syst. Sci.*, 2012

[12] S.Bocker, S. Briesemeister, Q.B.A. Bui, and A. Truss. Going weighted : Parameterized algorithms for cluster editing In *Theoretical Computer Science*, 2009

[13] J. Guo. A more effective linear kernelization for cluster editing In*Theoretical Computer Science*, 2009

[14] S. Bocker, S. Briesemeister and G. W. Klau. t Exact Algorithms for Cluster Editing : Evaluation and Experiments In*Algorithmica*, 2011

[15] M. L. Ginsberg and W. D. Harvey. Iterative Broadening In *Artificial Intelligence*, 1990

[16] C.T. Jun Zahn. Approximating symmetric relations by equivalence relations In *J. Soc. Ind. Appl. Math.*, 1964

[17] M.Sanchez, S. Bouveret, S. De Givry, F. Heras, P. Jegou, J. Larrosa, S. Ndiaye, S., E. Rollon, T. Schiex, C. Terrioux, G. Verfaillie and M. Zytnicki. Max-csp competition 2008 : toulbar2 solver description In *Proceedings of the Third International CSP Solver Competition*, 2008

[18] A. Ben-Dor, R. Shamir and Z. Yakhini. Clustering Gene Expression Patterns In*Journal of Computational Biology*, 1999

[19] E. Hartuv, A. Schmitt, J. Lange, S. Meier-Ewert, H. Lehrach, and R. Shamir. An algorithm for clustering cDNA fingerprints In *Genomics*, 2000

[20] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems In *Discrete Applied Mathematics*, 2004

[21] R. Sharan et al. CLICK and EXPANDER : a system for clustering and visualizing gene expression data In*Bioinformatics*, 2003

[22] T. Wittkop, J. Baumbach, F. Lobo, and S. Rahmann. Large scale clustering of protein sequences with FORCE–A layout based heuristic for weighted cluster editing In*BMC Bioinformatics*, 2007

[23] T. Schiex, H. Fargier and G. Verfaillie. Valued Constraint Satisfaction Problems : Hard and Easy Problems In*Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995

[24] S. Givry and M. Zytnicki. Existential arc consistency : Getting closer to full arc consistency in weighted csps In*Proc. of the 19 th IJCAI*, 2005

Une métaheuristique pour les problèmes de packing orthogonal en deux dimensions

S. Grandcolas[1] **C. Pain-Barre**[1]

[1]LSIS – UMR CNRS 7296 Av. Escadrille Normandie-Niemen, F-13397 Marseille Cedex 20 - France
{stephane.grandcolas,cyril.pain-barre}@lsis.org

Résumé

Nous présentons dans cet article une métaheuristique de type recherche locale pour les problèmes de packing orthogonal dans un espace à deux dimensions (OPP-2). Le processus, basé sur l'approche de F. Clautiaux et al. [8], consiste à chercher d'abord les positions des objets sur l'axe horizontal. Chaque fois qu'un placement satisfaisant est rencontré, une autre procédure tente de trouver les positions des objets sur l'axe vertical, de façon à ce qu'il n'y ait pas de chevauchements. Nous avons développé deux métaheuristiques distinctes, l'une pour la recherche des positions horizontales et l'autre pour la recherche des positions verticales, en envisageant différentes stratégies. Ces métaheuristiques peuvent facilement être utilisées pour traiter des problèmes de strip packing (SPP), et nous avons implémenté deux approches différentes pour SPP. Nous avons comparé nos résultats avec ceux obtenus par les systèmes les plus performants sur une sélection de problèmes connus.

Abstract

In this paper we describe a local search metaheuristic for solving orthogonal packing problems in a two-dimensional space (OPP-2). The method, based on F. Clautiaux et al. approach [8], consists first in searching the positions of the items on the horizontal axis. Each time a satisfying placement is discovered, another procedure searches the positions of the items on the vertical axis, so as that no two items overlap. We propose two distinct metaheuristics, one to search the positions of the items on the horizontal axis, and the other to search the positions of the items on the vertical axis. Several strategies have been developped and tested. These metaheuristics can easily be used to solve the strip packing problem (SPP), and we implemented two approaches for SPP. We compared our results with those obtained by the most efficient systems on a selection of SPP instances.

1 Introduction

Les problèmes de packing constituent une grande famille de problèmes dans le domaine de la recherche opérationnelle (bin packing, packing orthogonal, strip-packing, sac à dos en deux dimensions,...). Nous nous intéresserons dans cet article aux problèmes de packing en deux dimensions. Ces problèmes ont fait l'objet de nombreux travaux, du fait que leur description est très simple, et qu'ils correspondent à des problèmes réels très courants. Le problème de packing orthogonal (OPP) est le plus simple : il consiste à déterminer si n objets rectangulaires de largeurs $w_1, \dots w_n$ et de hauteurs $h_1, \dots, h_n$ peuvent être placés à l'intérieur d'une boîte rectangulaire de largeur W et de hauteur H sans qu'il y ait de chevauchements entre les objets. Le problème de strip packing (SPP) est plus connu : il consiste à placer un ensemble d'objets rectangulaires dans une bande de largeur donnée mais de hauteur infinie, de façon à ce que les objets ne se chevauchent pas et que la hauteur occupée dans la bande soit la plus petite possible. Nous considérerons dans cet article que les objets ont une orientation fixe pour ces deux problèmes (on trouve dans la littérature des méthodes qui envisagent la possibilité de rotation des objets).

SPP est un problème NP-difficile. Il est très lié au problème OPP. En effet, il est toujours possible de traiter SPP en testant successivement des problèmes OPP de hauteurs différentes. Les approches exactes les plus connues pour OPP sont la procédure de branch and bound de S. Martello et al. [14], la méthode de S. P. Fekete and J. Schepers [9] basée sur une représentation des placements des objets sur chaque axe à l'aide de graphes d'intervalles, et l'approche de F. Clautiaux et al. [8] dans laquelle on cherche d'abord les positions des objets sur l'axe horizontal avant de chercher leurs positions sur l'axe vertical. Ces approches ont des coûts élevés qui ne permettent pas de traiter des problèmes de taille importante. Il existe de nombreuses mé-

thodes à base d'heuristiques pour le problème SPP. Certaines utilisent l'algorithme *Bottom left* de B.S. Baker et al. [2] ou l'algorithme *Bottom left fill* de B. Chazelle [6]. Ces algorithmes placent les objets dans la boîte les uns après les autres de gauche à droite en commençant par le bas de la bande. Le placement obtenu dépend uniquement de l'ordre dans lequel sont pris les objets. Ainsi de nombreuses métaheuristiques utilisent l'algorithme BL en modifiant à chaque itération l'ordre des objets afin d'améliorer le placement. On en trouve à base de recherche tabou (M. Iori et al. [12]) ou d'algorithmes génétiques, ou encore qui modifient l'ordonnacement aléatoirement en se basant sur des probabilités (N. Lesh et al. [13]).

L'algorithme *Best Fit* de E.K. Burke et al. [5] est très différent de l'approche Bottom Left. A chaque étape on choisit l'objet qui convient le mieux pour être placé dans l'espace libre disponible le plus bas. R. Alvarez-Valdes et al. [1] proposent de construire une première solution de façon similaire à BF puis de l'améliorer avec un processus aléatoire de construction de type GRASP (Greedy Randomized Adaptive Search Procedure). Enfin, récemment B. Neveu et al. [15] ont décrit une approche originale de type recherche locale qui s'attache à modifier le placement des objets dans la boîte à partir des *trous*.

Nous proposons une nouvelle métaheuristique pour le problème OPP, basée sur l'approche exacte de F. Clautiaux et al., en utilisant une recherche locale. Cette métaheuristique peut être utilisée de deux façons différentes pour les problèmes de strip packing. La première consiste en une simple recherche dichotomique sur la hauteur. La deuxième consiste à modifier la recherche locale lors de la recherche des positions sur l'axe horizontal, en diminuant la hauteur au fur et à mesure que de nouveaux placements sont découverts. Nous avons implémenté ces deux approches et nous les avons comparées avec des méthodes performantes sur des jeux de problèmes classiques.

Nous présentons la métaheuristique pour le traitement du problème OPP dans la partie 2. La partie 3 est consacrée à l'adaptation de la métaheuristique au problème SPP. Les résutats de nos expérimentations figurent dans la partie 4. Enfin nous concluons dans la 5e partie.

2 Une métaheuristique pour OPP

L'approche de F. Clautiaux et al. [8] consiste à chercher les positions des objets sur l'axe horizontal (c'est à dire dans la largeur de la boîte), puis à déterminer les positions des objets sur l'axe vertical (c'est à dire dans la hauteur de la boîte). Leur procédure de recherche énumère tous les placements des objets sur l'axe des x qui pourraient correspondre à une solution du problème de packing. Lors de cette recherche, qualifiée de *phase externe*, chaque fois qu'un placement satisfaisant P_x est découvert, une autre procédure de recherche est appelée afin de déterminer s'il existe un placement P_y des objets sur l'axe vertical qui constitue avec P_x une solution du problème de packing. La recherche de P_y est appelée la *phase interne*.

La figure 1 reproduit une solution d'un problème OPP. Les placements P_x et P_y sont représentés par des ensembles d'intervalles figurant en-dessous de la boîte pour P_x et sur la gauche de la boîte pour P_y. Ils correspondent à la *projection* des objets sur l'axe des x pour P_x ou sur l'axe des y pour P_y. Ainsi la longueur des intervalles dans P_x correspond à la largeur des objets, tandis que la longuer des intervalles dans P_y correspond à leurs hauteurs. Le couple (P_x, P_y) constitue une solution du problème de packing si tous les intervalles sont contenus dans la largeur de la boîte pour P_x ou dans sa hauteur pour P_y, et s'il n'existe pas deux objets dont les intervalles ont une intersection non vide dans P_x et dans P_y. En effet, si c'était le cas, il y aurait un chevauchement entre ces deux objets dans la boîte.

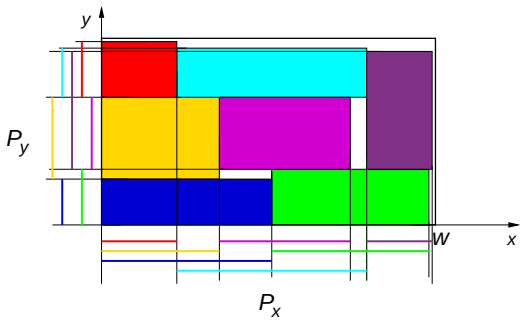


FIGURE 1 – Solution du problème OPP

Remarquons que si (P_x, P_y) est une solution du problème de packing, alors en tout point p de l'axe des x la somme des hauteurs des objets dont l'intervalle correspondant dans P_x contient p est nécessairement inférieure ou égale à la hauteur de la boîte. Lors de la recherche de P_x, il est donc inutile de considérer les placements qui n'ont pas cette propriété. Nous appellerons cette contrainte la *contrainte de hauteur*. Elle constitue une condition nécessaire pour que P_x puisse être étendu à une solution du problème de packing. Cette vérification (pour les largeurs) est inutile lorsqu'on cherche P_y. En effet, puisque à ce moment là les positions des objets sont fixées dans P_x, et puisque les intervalles correspondant à deux objets i et j ne doivent pas se chevaucher à la fois dans P_x et dans P_y, un ensemble I d'objets dont les intervalles s'intersectent deux à deux dans P_y n'admet aucune intersection dans P_x. La somme des largeurs des intervalles correspondant aux objets de I dans P_x est donc inférieure ou égale à la largeur W de la boîte.

L'approche de F. Clautiaux et al. est exacte : toutes les configurations possibles sont explorées dans la phase externe, et pour chaque placement P_x qui satisfait la contrainte de hauteur, une procédure explore toutes les façons pos-

sibles de contruire P_y. Récemment S. Grandcolas et al. [10] ont présenté des améliorations de cette approche, qui permettent de réduire l'espace de recherche. Cependant, la taille des problèmes qui peuvent être traités en temps raisonable reste très limitée. Nous proposons dans cet article d'implémenter l'approche de F. Clautiaux et al. en calculant les placements des objets en phase externe et en phase interne à l'aide de métaheuristiques. La recherche est alors incomplète, mais il devient possible de traiter des problèmes de tailles importantes. Nous décrivons ci-dessous deux métaheuristiques pour la recherche de P_x et de P_y, basées sur des recherches locales.

Recherche du placement P_x (phase externe)

Les positions des objets dans P_x (ainsi que dans P_y) sont comprises entre 0 et l'infini, la position 0 correspondant à la face gauche de la boîte (ou au bas de la boîte pour P_y). Ainsi les objets peuvent "dépasser" à droite mais pas à gauche (ou en haut mais pas en bas dans P_y). La figure 2 illustre la recherche du placement P_x lors de la phase externe. Nous avons ajouté au-dessus des intervalles une représentation en deux dimensions de l'espace vertical occupé par les objets en chaque point de l'axe des x (c'est à dire la somme des hauteurs des objets qui recouvrent ce point). Dans cette représentation chaque objet apparaît à la position sur l'axe des x qu'il occupe dans P_x. Les objets sont empilés les uns sur les autres dans un ordre arbitraire, et éventuellement découpés en plusieurs parties afin qu'il n'y ait aucun espace libre. Le rectangle grisé rappelle les dimensions de la boîte.

La recherche d'un placement P_x satisfaisant consiste, à partir de positions des intervalles choisies aléatoirement, à appliquer des modifications élémentaires dans le but de satisfaire la contrainte d'inclusion (les intervalles doivent être contenus dans la largeur de la boîte) et la contrainte de hauteur. Une modification élémentaire se décompose en trois étapes : (1) enlever un intervalle du placement, (2) réarranger éventuellement le placement, et (3) réinsérer l'intervalle dans le placement. En fait, lors de l'insertion d'un intervalle i dans le placement, seules sont considérées la position 0 (la face gauche de la boîte) et les positions correspondant à des bornes droites d'intervalles déja dans le placement (nous qualifierons ces intervalles de *supports* de l'intervalle i). Cette restriction n'a aucun effet sur l'issue de la recherche, si le problème de packing a des solutions alors il en existe de ce type. Remarquons enfin qu'un intervalle peut perdre tous ses supports pendant la recherche. Nous avons envisagé de réarranger P_x chaque fois qu'un intervalle est enlevé, afin que tous les intervalles aient au moins un support à tout moment, en *tassant* le placement vers la gauche chaque fois que c'est nécessaire (en déplaçant les intervalles non soutenus vers la gauche, et en propageant ces décalages). Cette option est implémentée mais ne produit pas de meilleurs résultats sur les instances que nous avons traitées.

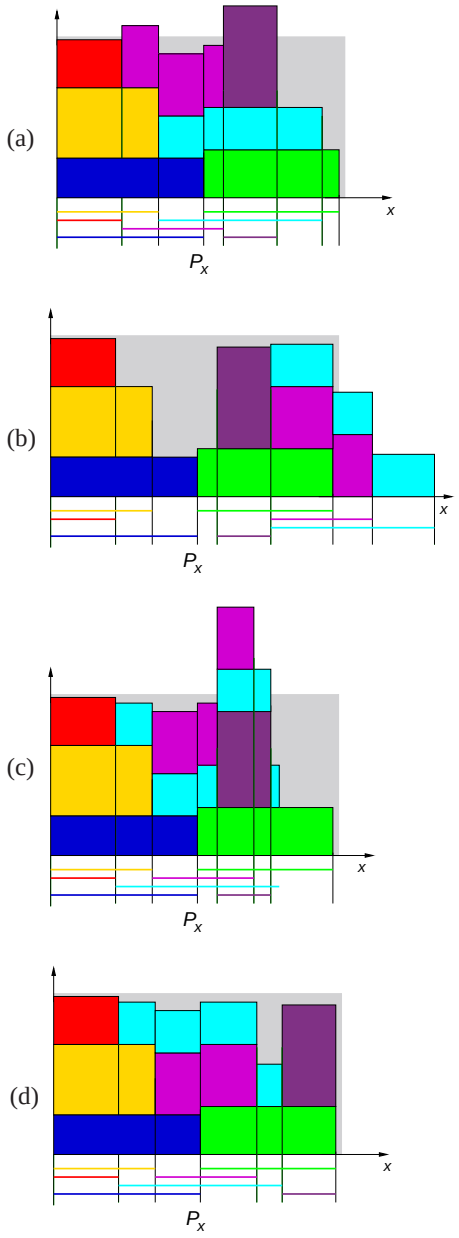


FIGURE 2 – Phase externe : contraction et écrasement

La procédure de recherche alterne des périodes de *contraction* et des périodes d'*écrasement* du placement P_x, tant que les contraintes d'inclusion et de hauteur ne sont pas satisfaites, et que le nombre de modifications effectuées est en-dessous d'une limite donnée. Chaque période a un objectif différent. Le changement de période intervient quand l'objectif de la période est atteint, ou quand le

nombre de modifications effectuées durant la période courante dépasse une limite donnée (fixée à 10 pour nos expérimentations).

Contraction. L'objectif est de placer les intervalles de P_x dans la largeur de la boîte. On choisit des intervalles qui *dépassent* à droite, et on les insère dans la largeur de la boîte, quitte à générer ou à renforcer une violation de la contrainte de hauteur (voir la transition de (b) à (c) dans la figure 2 : l'objet bleu turquoise et l'objet rose, qui violent la contrainte d'inclusion, sont déplacés vers la gauche de façon à ce que les intervalles correspondants soient contenus dans la largeur de la boîte). S'il en existe, on choisit comme destination pour l'intervalle déplacé une position qui ne produit pas de dépassement de la hauteur, en privilégiant les positions qui maximisent l'espace libre dans la bande verticale délimitée par l'intervalle. Si il n'existe pas de telles positions, on privilégiera les positions qui minimisent la surface exédentaire.

Ecrasement. Le but est de satisfaire la contrainte de hauteur : les intervalles choisis pour être déplacés sont des intervalles impliqués dans la violation de la contrainte de hauteur. Nous avons implémenté différents choix pour les destinations des intervalles déplacés. Les meilleurs résultats ont été obtenus en choisissant une destination telle que la contrainte de hauteur soit satisfaite en tout point de l'intervalle déplacé, en privilégiant celles qui maximisent la hauteur occupée au niveau de la borne gauche de l'intervalle. Dans la figure 2, la transition de (a) à (b) avec le déplacement de l'objet rose et de l'objet bleu turquoise, et la transition de (c) à (d) avec le déplacement de l'objet violet illustrent la phase d'écrasement. En (d) le placement P_x satisfait la contrainte de hauteur et la contrainte d'inclusion.

Recherche du placement P_y (phase interne)

La recherche du placement P_y est assez semblable à la recherche de P_x, modulo le fait que, puisque les positions des objets sur l'axe des x sont déja fixées, deux intervalles ne peuvent se chevaucher dans P_y si les intervalles correspondant à ces mêmes objets dans P_x ont une intersection non vide. Nous dirons que deux objets sont en *conflit* si les intervalles correspondant à ces objets dans P_x et dans P_y s'intersectent dans les deux placements. Nous avons pris le parti de considérer, pendant la recherche de P_y, uniquement des placements sans conflit. Afin d'éviter l'apparition de conflits chaque fois qu'un intervalle est inséré dans P_y, les intervalles qui sont en conflit avec cet intervalle sont décalés vers le haut (jusqu'à la borne haute de l'intervalle). Un mécanisme de propagation prend en charge ces décalages qui peuvent à leur tour générer des conflits. Le processus s'arrête lorsqu'il n'y a plus de conflits dans P_y.

La recherche de P_y consiste, partant d'un placement aléatoire mais sans conflit, à déplacer des intervalles en éliminant chaque fois les conflits, dans le but de faire rentrer tous les intervalles de P_y dans la hauteur de la boîte. Les intervalles déplacés sont choisis parmis ceux qui dépassent le sommet de la boîte. Les destinations possibles sont la position 0 (qui correspond au bas de la boîte) et les bornes supérieures des intervalles du placement. La recherche de P_y prends fin dès que tous les intervalles de P_y sont contenus dans la hauteur, ou dès que le nombre de déplacements effectués atteint une limite donnée.

Nous avons retenu les options suivantes pour le choix des déplacements :

lowest increase width On considère le déplacement d'un intervalle i à la position p seulement si (1) i est *supporté* par un objet en conflit à la position p (i.e. un intervalle correspondant à un objet qui recoupe l'objet correspondant à i dans P_x se termine en p ; en deux dimensions cela signifie que l'objet correspondant à i sera soutenu par un objet situé en-dessous) et si (2) il n'y a pas d'intervalle en conflit avec i qui recouvre la position p. Cette dernière condition signifie que pour résoudre les conflits il n'y aura pas à déplacer des intervalles débutant avant p. On favorise parmi les déplacements qui satisfont ces deux conditions ceux dont la destination est la plus basse, et pour lesquel la largeur occupée au niveau de la destination augmente le plus (l'augmentation correspond à la différence entre la largeur de l'objet et la somme des largeurs des objets en conflit qui seront décalés vers le haut).

minimize area/minimize height Choix d'une destination telle que l'aire en conflit (ou la hauteur cumulée des conflits) est minimale.

random Choix d'un objet et d'une position au hasard. Si c'est possible, on choisit une position sans conflit.

Pour nos expérimentations nous avons utilisé dans 10 pour cent des cas le choix **random**, sinon le choix **lowest increase width**, et s'il n'y a pas de candidats le choix **minimize area/minimize height**.

3 Problèmes de Strip Packing

Le problème de strip packing (SPP) consiste à trouver la hauteur minimale qui permet de placer n objets dans une bande de largeur donnée l. De nombreuses méthodes complètes ou non ont été publiées pour ce problème. Nous proposons deux approches. La première consiste en une recherche dichotomique sur la hauteur : la fonction SPP-dicho calcule la hauteur minimale d'une bande de largeur l dans laquelle on doit placer l'ensemble d'objets O. Initialement la plage de recherche correspond à des bornes inf et sup triviales. La borne sup est obtenue avec un algorithme connu, le *shelf algorithm*, qui consiste à poser les objets les uns après les autres sur un même niveau de la gauche vers la droite (comme sur une étagère d'où le

nom *shelf algorithm*), en ouvrant un nouveau niveau dès qu'il n'y a plus assez d'espace pour poser l'objet suivant. La borne inf correspond simplement à la surface totale des objets divisée par la largeur de la boîte l. Tant que la plage $[inf, sup]$ contient plus d'une valeur, une hauteur intermédiaire h est testée à l'aide de la procédure décrite dans la section précédente pour le problème OPP. Dans notre implémentation nous avons pris comme hauteur intermédiaire la moyenne des bornes inf et sup. Initialement, le placement P_x (resp. P_y) est généré aléatoirement par la fonction genere_$P_x()$ (resp. genere_$P_y()$). La fonction ameliore_$P_x()$ (resp. ameliore_$P_y()$) tente d'améliorer P_x (resp. P_y) tant qu'il n'est pas valide, en effectuant au plus $nSteps_x$ (resp. $nSteps_y$) modifications, comme cela a été décrit dans la partie précédente pour le problème OPP. En phase externe le placement P_x doit satisfaire la contrainte d'inclusion et la contrainte de hauteur, tandis qu'en phase interne le placement P_y est astreint à respecter la contrainte d'inclusion et la contrainte de non-chevauchement définie par le graphe d'intersection des intervalles de P_x, noté G_x.

Si la procédure trouve une solution pour la hauteur h on diminue la borne sup, sinon on augmente la borne inf (on prendra par exemple la moyenne entre les bornes inf et sup). Dans notre implémentation, si le placement P_x est valide, la recherche de P_y est réitérée un certain nombre de fois, en changeant les paramètres afin d'intensifier la recherche, tant qu'on ne trouve pas un placement valide. Enfin, la fonction SPP-dicho peut être appelée plusieurs fois, en mettant à jour la borne sup avec le meilleur résultat rencontré lors des essais précédents.

La deuxième procédure que nous avons développée pour SPP, nommée SPP*, détermine une valeur minimale de la hauteur lors de la recherche du placement P_x dans la phase externe (fonction optimise_P_x). Initialement, la valeur de la hauteur pour la contrainte de hauteur est $sup - 1$, où sup est la borne produite par le *shelf algorithm*. Chaque fois qu'un placement P_x satisfaisant la contrainte d'inclusion et la contrainte de hauteur est découvert, la fonction optimise_P_x réitère la recherche avec la valeur de la hauteur diminuée d'une unité. Ainsi la hauteur diminue jusqu'à ce que la recherche locale soit incapable de produire un placement satisfaisant les contraintes d'inclusion et de hauteur. Le placement P_x retenu pour la phase interne est le dernier placement valide rencontré pendant cette phase d'amélioration, c'est donc celui dont la hauteur maximale occupée h est la plus petite. Si la recherche du placement P_y est infructueuse, la fonction SPP* renvoie comme résultat la borne sup produite par le *shelf algorithm*.

Réglage des paramètres. Pour chaque procédure les paramètres $nSteps_x$ et $nSteps_y$ sont choisis en fonction de la taille du problème (c'est à dire du nombre de variables). La procédure est exécutée plusieurs fois, et chaque fois les paramètres sont durcis. L'idée est de renforcer la recherche au fur et à mesure que la plage de recherche de la hauteur se restreint. Il est en effet inutile d'effectuer des recherches coûteuses lors des premiers essais, alors que les hauteurs testées peuvent être très éloignées de la hauteur optimale, et la recherche très facile ou au contrainte sans solution. Le temps de calcul est à peu près doublé entre deux essais successifs (le nombre d'étapes augmente tandis que le nombre d'essais diminue).

Function SPP-dicho(O, l, $nSteps_x$, $nSteps_y$)

```
begin
    sup := borne_sup(O, l);
    RESTART :
    inf := borne_inf(O, l);
    while inf < sup do
        h := (inf + sup)/2;
        P_x := genere_P_x(O);
        P_x := ameliore_P_x(P_x, O, l, h, nSteps_x);
        if valide_P_x(P_x, l, h) then
            G_x := calcul_chevauchements(P_x);
            P_y := genere_P_y(O, G_x);
            P_y := ameliore_P_y(P_y, O, h, G_x, nSteps_y);
            if valide_P_y(P_y, G_x, h) then
                sup := h;
            else
                inf := h;
    if not time_out then
        if sup ≠ borne_inf(O, l) then
            goto RESTART;
    return sup;
end
```

Function SPP*(O, l, $nSteps_x$, $nSteps_y$)

```
begin
    P_x := genere_P_x(O);
    (P_x, h) := optimise_P_x(P_x, O, l, nSteps_x);
    G_x := calcul_chevauchements(P_x);
    P_y := genere_P_y(O, G_x);
    P_y := ameliore_P_y(P_y, O, h, G_x, nSteps_y);
    if valide_P_y(P_y, G_x, h) then
        return h;
    else
        return borne_max(O, l);
end
```

Terminaison. La recherche s'arête après un certain nombre d'essais, ou lorsque la meilleure hauteur trouvée n'évolue plus pendant plusieurs essais successifs, ou encore lorsque qu'un temps limite donné est dépassé.

Remarque sur le coût. Le temps de calcul de la fonction SPP-dicho est fonction de la taille du problème (le nombre

de variables détermine le nombre d'essais et le nombre d'étapes), mais aussi de sa *granularité*. En effet, le nombre d'exécutions de la boucle dépends de la largeur de la plage de recherche : si inf et sup sont les bornes initiales, le nombre d'appels est $O(\log\ (sup - inf + 1))$ comme pour une recherche dichotomique.

4 Résultats expérimentaux

Nous avons comparé les procédure de recherche SPP-dicho et SPP* avec des approches récentes et performantes, sur des séries de problèmes classiques (avec orientation fixe des objets). Nous avons choisi de reporter les résultats de GRASP de R. Alvarez-Valdes et al. [1] et du système de B. Neveu et al. [15] (l'option retenue est ID Walk avec l'heuristique HH et le réarrangement P, qui semble donner les meilleurs résultats). Les résultats figurent dans la table 1. Nous nous sommes limité aux jeux d'instances *gcut* de J.E. Beasley [3], *cgcut* de N. Christofides et al. [7], et *beng* proposées par B.E. Bengtsson [4]. Il s'agit de problèmes *généraux*, dans le sens où les solutions optimales ne sont pas toujours connues, et sont susceptibles de contenir de l'espace vide. Pour chaque instance figurent la moyenne des hauteurs et la hauteur minimale trouvée par chaque solveur sur une suite d'essais (10 essais avec un temps limite de 60 secondes pour GRASP et 10 essais avec un temps limite de 100 secondes pour ID Walk).

Nous avons reporté les résultats que nous avons obtenu en lançant nos procédures de recherche 12 fois pour chaque instance avec un temps limite de 1000 secondes pour chaque essai (ce temps limite a été atteint uniquement avec les instances les plus difficiles) sur des machines équipées de processeurs Xeon 2.4 GHz. Les paramètres de SPP-dicho et SPP* sont fixés au lancement du programme. Ils dépendent uniquement du nombre de variables du problème (au delà de 50 variables les paramètres n'évoluent plus). Pour certains problèmes, *cgcut*3 par exemple, il arrive que la moyenne des résultats soit fortement dégradée du fait que certains essais ont terminé sans produire de résultat, et nous avons alors utilisé la borne supérieure correspondant à un placement trivial des objets. C'est un défaut de SPP* de terminer sans avoir établi de résultat : en général la phase 1 produit très rapidement un placement P_x de bonne qualité (de hauteur très petite). La recherche du placement P_y en phase interne est souvent fastidieuse, et peut même ne jamais aboutir avant le temps limite. Cela explique les mauvais résultats de SPP*, qui n'est jamais meilleur que SPP-dicho.

Pour chaque série de problèmes figurent les moyennes des résultats moyens et des meilleurs résultats. Pour les problèmes *gcut* nous n'avons pas intégré dans ces moyennes les résultats des instances *gcut*9,...,*gcut*13 (indisponibles pour GRASP).

Sur les instances *gcut*, SPP-dicho est meilleur que ses concurrents (exception faite de l'instance *gcut*13 pour laquelle la hauteur minimale trouvée par ID Walk est inférieure). Les résultats qui constituent une amélioration significative apparaissent en gras. C'est le cas notamment pour les instances *gcut*4, *gcut*8, *gcut*13*r* qui sont difficiles. Ses moyennes sur la série sont meilleures que celles de GRASP et ID Walk. Inversement, sur la série *beng* SPP-dicho n'est pas performant, surtout sur les instances dont le nombre de variables est supérieur à 60. Nous pourrions faire les mêmes observations sur les instances de Hopper et Turton [11]. Dans ce cas les solutions optimales sont connues et ne laissent aucun espace vide dans la boîte. D'autre part, il y a très peu de diversité dans les dimensions des objets pour les instances *beng* (pour *beng*4 par exemple on compte 80 objets et seulement 8 largeurs différentes, alors que pour *gcut*8 on compte 50 objets avec 44 largeurs différentes). Cela conduit à des placements dans lesquels on trouve souvent des objets de même largeur (resp. hauteur) placés côte à côte qui constituent des agrégats rectangulaires. Notre méthode ne tient pas compte de ce types de configurations (GRASP par exemple tente de constituer des blocs plaçant côte à côte plusieurs pièces de mêmes dimensions). Il semble que SPP soit plus efficace lorsque les pièces sont peu nombreuses, avec des dimensions rarement égales mais assez proches (peu dispersées). D'autre part les modifications du placement P_y dans la phase interne ont tendance à bouleverser le placement courant. L'insertion d'un objet produit le décalage des objets en conflits, décalages qui se propagent dans le placement et qui peuvent le changer de façon importante. Ce comportement a tendance à freiner la convergence vers un placement valide.

5 Conclusion

Nous avons présenté une métaheuristique pour les problèmes de packing orthogonal en deux dimensions basé sur l'approche (exacte) de F. Clautiaux [8]. Nous avons implémenté la méthode présentée, et nous l'avons adaptée au traitement de problèmes de Strip Packing. Les deux procédures de recherche SPP-dicho et SPP* que nous avons développées ont été comparées avec des approches existantes qui font référence sur des jeux d'instances classiques. Les résultats sont bons sur les instances dans lesquelles les objets sont peu semblables mais leurs dimensions assez proches.

Nous allons chercher à rendre plus efficace la recherche locale de la phase 2, en introduisant de nouvelles modifications élémentaires et en privilégiant celles qui ne modifient pas le placement de façon importante.

Références

[1] R. Alvarez-Valdes, F. Parreño, and J.M. Tamarit. Reactive grasp for the strip-packing problem. *Com-*

Instance				GRASP		IDW (+HH+P)		SPP-dicho		SPP*	
Name	*n*	*w*	*LB*	*moy.*	*min*	*moy.*	*min*	*moy.*	*min*	*moy.*	*min*
gcut1	10	250	1016	1016,0	1016	1016,0	1016	1016,0	1016	1016,0	1016
gcut2	20	250	1133	1191,0	1191	1203,9	1195	1187,0	1187	1191,1	1187
gcut3	30	250	1803	1803,0	1803	1803,0	1803	1803,0	1803	1803,0	1803
gcut4	50	250	2934	3002,0	3002	3032,0	3020	3003,7	**3000**	3009,1	3004
gcut5	10	500	1172	1273,0	1273	1273,0	1273	1273,0	1273	1273,0	1273
gcut6	20	500	2514	2627,0	2627	2651,1	2639	2622,0	2622	2622,0	2622
gcut7	30	500	4641	4693,0	4693	4710,1	4704	4693,0	4693	4693,0	4693
gcut8	50	500	5703	5912,2	5908	5947,7	5895	**5877,6**	**5872**	5934,9	5890
gcut9	10	1000	2022			2317,0	2317	2317,0	2317	2317,0	2317
gcut10	20	1000	5356			5972,0	5969	5964,0	5964	5964,0	5964
gcut11	30	1000	6537			6994,1	6966	6869,5	6866	6894,8	6875
gcut12	50	1000	12522			14690,0	14690	14690,0	14690	14690,0	14690
gcut13	32	3000	4772			4975,7	4914	4953,4	4932	5133,8	4992
gcut9r	10	1000	2022	2256,0	2256	2251,3	2241	2241,0	2241	2241,0	2241
gcut10r	20	1000	5356	6393,0	6393	6427,3	6422	6393,0	6393	6393,0	6393
gcut11r	30	1000	6537	7736,0	7736	7736,0	7736	7736,0	7736	7736,0	7736
gcut12r	50	1000	12522	13172,0	13172	13213,0	13172	13172,0	13172	13172,0	13172
gcut13r	32	3000	4772	5009,5	5009	5075,4	5028	5015,5	**5007**	5070,7	5028
				4314,1	4313,7	4333,8	4318,7	4310,2	**4308,8**	4319,6	4312,1
cgcut1	16	10	23	23,0	23	23,0	23	23,0	23	23,0	23
cgcut2	23	70	63	65,0	65	65,0	65	64,9	64	65,1	65
cgcut3	62	70	636	661,0	661	667,9	662	662,0	661	721,9	663
beng01	20	25	30	30,0	30	30,4	30	30,0	30	30,0	30
beng02	40	25	57	57,0	57	58,0	58	58,0	58	58,5	58
beng03	60	25	84	84,0	84	84,5	84	88,2	87	92,0	86
beng04	80	25	107	107,0	107	107,9	107	119,4	116	126,0	126
beng05	100	25	134	134,0	134	134,0	134	152,7	149	153,0	153
beng06	40	40	36	36,0	36	36,0	36	36,0	36	36,0	36
beng07	80	40	67	67,0	67	67,3	67	74,9	73	76,6	72
beng08	120	40	101	101,0	101	101,0	101	109,0	109	109,0	109
beng09	160	40	126	126,0	126	126,0	126	140,0	140	140,0	140
beng10	200	40	156	156,0	156	156,0	156	171,0	171	171,0	171
				89,8	89,8	90,1	89,9	97,9	96,9	99,2	98,1

TABLE 1 – Hauteurs moyennes et hauteurs minimales (SPP-2)

puters & Operations Research, 35(4) :1065 – 1083, 2008.

[2] B. Baker, E. Coffman, and R. Rivest. Orthogonal packing in two dimensions. *SIAM J. of Computing*, 9(4) :846–855, 1980.

[3] J. E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *The Journal of the Operational Research Society*, 36(4) :297–306, 1985.

[4] B.E. Bengtsson. Packing rectangular pieces - a heuristic approach. *Comput. J.*, 25(3) :353–357, 1982.

[5] E. K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Oper. Res.*, 52(4) :655–671, August 2004.

[6] B. Chazelle. The bottom-left bin-packing heuristic : An efficient implementation. *IEEE Transactions on Computers*, 32(8) :697–707, 1983.

[7] N. Christofides and C. Whitlock. An Algorithm for Two-Dimensional Cutting Problems. *Operations Research*, 25(1) :30–44, January 1977.

[8] F. Clautiaux, J. Carlier, and A. Moukrim. A new exact method for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183(3) :1196–1211, 2007.

[9] S. P. Fekete and J. Schepers. A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research*, 29(2) :353–368, 2004.

[10] S. Grandcolas and C. Pinto. A new search procedure for the two-dimensional orthogonal packing problem. *Journal of Mathematical Modelling and Algorithms in Operations Research*, 14(3) :343–361, 2015.

[11] E. Hopper and C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research*, 128(1) :34–57, 2001.

[12] Manuel Iori, Silvano Martello, and Michele Monaci. *Optimization and Industry : New Frontiers*, chapter Metaheuristic Algorithms for the Strip Packing Pro-

blem, pages 159–179. Springer US, Boston, MA, 2003.

[13] N. Lesh, J. Marks, A. McMahon, and M. Mitzenmacher. New heuristic and interactive approaches to 2d rectangular strip packing. *J. Exp. Algorithmics*, 10, December 2005.

[14] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *Journal on Computing*, 15(3) :310–319, 2003.

[15] B. Neveu, G. Trombettoni, I. Araya, and M.C. Riff. A strip packing solving method using an incremental move based on maximal holes. *International Journal on Artificial Intelligence Tools*, 17(5) :881–901, 2008.

Complexité paramétrée de l'optimisation sous contrainte DNNF

Frédéric Koriche Daniel Le Berre Emmanuel Lonca Pierre Marquis

Université d'Artois, CRIL-CNRS UMR 8188
{koriche,leberre,lonca,marquis}@cril.fr

Résumé

La minimisation de fonction de coût sous contraintes combinatoires est un problème fondamental en IA, bien que difficile à résoudre. Dans un certain nombre d'applications réelles, l'ensemble des contraintes décrivant la structure du problème évolue moins souvent que la fonction de coût capturant les préférences de l'utilisateur. Dans de telles situations, il peut être intéressant de compiler l'ensemble des solutions réalisables en amont, particulièrement lorsque le langage cible permet de rendre moins difficile le calcul d'une solution optimale pour des fonctions de coût connues après la phase de compilation. Dans cet article, nous étudions la complexité du problème de minimisation sous contrainte DNNF pour différentes familles de fonctions de coût. Au-delà du cas linéaire déjà connu comme traitable, nous montrons que la minimisation des fonctions quadratiques et des fonctions sous-modulaires sont *traitables à paramètre fixé* pour différents sous-ensembles de DNNF. Plus précisément, en ce qui concerne la minimisation sous contrainte des fonctions sous-modulaires, nous montrons que la *traitabilité à paramètre fixé* est établie pour un paramètre k naturel défini à partir de la dissimilarité structurelle entre la fonction de coût et la formule DNNF.

Abstract

Minimizing a cost function under a set of combinatorial constraints is a fundamental, yet challenging problem in AI. Fortunately, in various real-world applications, the set of constraints describing the problem structure is much less susceptible to change over time than the cost function capturing user's preferences. In such situations, compiling the set of feasible solutions during an offline step can make sense, especially when the target compilation language renders computationally easier the generation of optimal solutions for cost functions supplied "on the fly", during the online step. In this paper, the focus is laid on Boolean constraints compiled into DNNF representations. We study the complexity of the minimization problem for several families of cost functions subject to DNNF constraints. Beyond linear minimization which is already known to be tractable in the DNNF language, we show that both quadratic minimization and submodular minization are *fixed-parameter tractable* for various subsets of DNNF. In particular, the fixed-parameter tractability of constrained submodular minimization is established using a natural parameter capturing the structural dissimilarity between the submodular cost function and the DNNF representation.

1 Introduction

L'optimisation sous contraintes est un problème fondamental en intelligence artificielle (IA), pour plusieurs applications comme la configuration, la prise de décision assistée par ordinateur, les systèmes de recommandation, ou le *e-commerce*. Pour beaucoup de ces applications, l'espace des solutions réalisables est de nature combinatoire ; dans le cadre de l'IA, on peut représenter cet espace de solutions implicitement, de façon compacte, en utilisant une approche générique, les *réseaux de contraintes valuées* [5, 24, 27, 32, 35]. De manière informelle, un réseau de contraintes valuées peut être vu comme l'association d'un ensemble de variables discrètes, d'une collection de contraintes *dures* encodant l'espace des solutions réalisables, et un ensemble de contraintes souples valuées (ou *potentiels*) permettant d'encoder des préférences sur les différentes solutions ; le problème est alors de déterminer une solution réalisable qui minimise la somme des potentiels. Cette structure particulière apparaît dans l'état de l'art sous différents noms, comme les *réseaux GAI* [1, 18] ou encore les *conditional random fields* [23, 29]. La richesse de ce formalisme a cependant un coût : le problème de minimisation associé à un réseau de contraintes valuées est NP-difficile, ce qui exclut toute garantie de performance à son sujet.

Cependant, dans un grand nombre d'applications réelles, l'ensemble des contraintes décrivant la structure du problème évolue peu, en comparaison à l'ensemble des contraintes souples capturant les préférences de l'utilisateur, qui ont tendance à changer en même temps

que l'utilisateur lui-même. Une telle situation peut être exploitée grâce à une approche de type *compilation de connaissances* [11] : on peut chercher à compiler l'ensemble des contraintes dures lors d'une phase en amont des processus d'optimisation, dans le but d'améliorer le temps de calcul nécessaire à la phase ultérieure d'optimisation, dépendant, elle, des préférences de l'utilisateur concerné.

Considérons par exemple un service web de configuration de maison. En raison de sa nature combinatoire, l'ensemble des résidences qu'il est possible de produire est représenté de manière implicite par un ensemble de contraintes dures, telles que *la maison doit posséder au moins deux chambres*, ou *seules les maisons qui ont au moins une grande chambre peuvent avoir une cuisine premium*. Il est évident que l'ensemble des solutions réalisables ne dépend nullement des exigences de l'utilisateur (*je veux une cuisine premium*) ou de leurs préférences (*je préfère les grandes chambres aux petites*) ; dans ce cas, il peut être souhaitable de compiler en amont les contraintes (dures) décrivant l'espace des solutions si cela permet par la suite de pouvoir réduire le temps de calcul d'une solution réalisable et non dominée une fois les préférences de l'utilisateur connues.

Dans cet article, les contraintes dures sont représentées par des circuits (booléens) sous forme `NNF`. Nous nous intéressons plus particulièrement aux contraintes compilées dans le langage des circuits `DNNF` [8], le sous-langage de `NNF` pour lequel les nœuds ET ne partagent pas de variables. Le langage `DNNF` est un des sous-langages les plus succincts de `NNF` pour lequel il existe un algorithme polynomial permettant de déterminer si une affectation partielle des variables peut être étendue en une solution réalisable. Cette propriété est naturellement préservée pour les sous-langages de `DNNF`, parmi lesquels on peut citer la classe `DNF` des formules sous forme normale disjonctive, la classe `SDNNF` des formules `DNNF` structurées [26], le langage `SDD` des *sentential decision diagrams*, ou encore le langage `OBDD` des diagrammes de décision binaires ordonnés [3]. D'un point de vue pratique, le choix du langage `DNNF` est motivé par l'existence de compilateurs produisant des formules appartenant à (un sous-ensemble de) ce langage, comme par exemple `c2d` [9], `sdd` [10], et `Dsharp` [25].

Dans la suite de l'article, nous considérons différentes familles $\mathcal{F}$ de fonctions de coût pseudo-booléennes, qui incluent en particulier les fonctions *sous-modulaires*, pour lesquelles l'intérêt est grand [16, 28, 20], dû notamment à leurs nombreuses applications en IA [1]. Le but de cet article est d'identifier la complexité de la requête de minimisation **MIN** : étant donné une contrainte dure C donnée par une formule d'un sous-ensemble $\mathcal{L}$ de `DNNF` et une fonction de coût f exprimée par une somme de potentiels d'une famille $\mathcal{F}$, trouver (s'il en existe une) une solution réalisable de C qui minimise f. Ainsi, dans le cadre des réseaux de contraintes valuées, notre but est déterminer si un problème de minimisation sous contraintes est traitable ou non lorsque les contraintes souples appartiennent à $\mathcal{F}$ et les contraintes dures ont été au préalable compilées en une unique contrainte C de $\mathcal{L}$. Ainsi, pour différents sous-ensembles $\mathcal{L}$ de `DNNF` et différentes familles $\mathcal{F}$ de fonctions de coûts, nous déterminons si le problème de minimisation défini sur $\mathcal{L}$ et $\mathcal{F}$ est dans P, ou bien s'il est NP-difficile.

En plus de cela, en nous appuyant sur les travaux de Downey et Fellows sur la complexité paramétrée [13], nous identifions parmi les cas NP-difficiles certaines restrictions pour lesquelles le problème est *traitable à paramètre fixé*. Dans la théorie de la complexité paramétrée, l'efficacité d'un algorithme est jugée en considérant deux mesures : l'habituelle taille des données d'entrée n, et un paramètre additionnel k, qui représente généralement une mesure structurelle sur les données d'entrée, comme par exemple la largeur d'arbre lorsque l'entrée est un graphe. Les algorithmes traitables à paramètre fixé (FPT, pour *fixed-parameter tractable*) sont ceux pour lesquels le temps d'exécution est de la forme $p(k)n^{\mathcal{O}(1)}$ pour une fonction p qui ne dépend que de k. Dans le formalisme que nous considérons, **MIN** est FPT pour le paramètre k si, pour toute contrainte C de $\mathcal{L}$, la minimisation de f sous contrainte C peut être réalisée en un temps polynomial dans les tailles de C et f, avec un polynôme dont le degré est indépendant de k. D'un point de vue pratique, l'existance d'un algorithme FPT pour la requête **MIN** indique que l'optimisation peut être réalisée de manière efficace lorsque les valeurs de k sont suffisamment petites, même lorsque les tailles du circuit C et de la représentation de la fonction de coût f sont importantes.

Le principal résultat connu pour la requête **MIN** est celui de la minimisation des fonctions linéaires sous contrainte `DNNF` (dans P)[12], qui correspond dans notre formalisme au cas où les potentiels sont réduits à des singletons. De ce fait, nous nous sommes intéressés à la complexité de **MIN** pour des familles de fonctions de coûts $\mathcal{F}$ plus générales, pour lesquelles la littérature apporte un certain nombre de résultats négatifs. Parmi ces résultats, on peut notamment citer le cas de la famille des fonctions quadratiques, exprimées comme une somme de potentiels d'arité au plus 2, dont la minimisation *non contrainte* est NP-difficile par réduction immédiate depuis le problème MIN-2-SAT [7, 33]. En ce qui concerne les fonctions sous-modulaires, il est bien connu que la minimisation sans contrainte est dans P [20, 21] ; cependant, la version sous contraintes est généralement NP-difficile[27], même dans le cas très restreint où l'ensemble des solutions réalisables est donné par une unique contrainte de cardinalité [30, 31].

Dans la continuité des travaux initiés par Darwiche et Marquis [12], notre contribution fournit un éventail de résultats plus large pour l'optimisation sous contraintes `DNNF`, en particulier pour ce qui concerne les fonctions

1. voir par exemple `submodularity.org/references.html`

sous-modulaires. D'un côté, nous montrons que **MIN** est NP-difficile à la fois pour les fonctions sous-modulaires quadratiques sous contraintes OBDD et pour les fonctions sous-modulaires générales sous contraintes DNNF structurées en arbres, ce qui implique directement que la minimisation de fonction sous-modulaire n'est pas traitable sous contrainte DNNF. D'un autre côté, nous montrons que **MIN** est FPT pour plusieurs sous-ensembles $\mathcal{L}$ de DNNF et familles $\mathcal{F}$ de fonctions sous-modulaires. Dans notre étude, le paramètre clé k qui assure la traitabilité est donné par une mesure de dissimilarité entre la structure de la fonction de coût et la structure de la formule DNNF. Pour résumer, le message de cet étude est que la *compatibilité structurelle* entre la contrainte dure et la fonction de coût joue un rôle prépondérant dans l'efficacité de l'optimisation sous contraintes.

2 Préliminaires formels

Nous débutons par une présentation de quelques concepts et notations qui sont utilisés tout au long de cet article. Étant donné un entier positif n, $[n]$ représente l'ensemble $\{1,...,n\}$. On note $\mathbb{Q}_+$ l'ensemble des nombres rationnels non négatifs, et on définit $\overline{\mathbb{Q}}_+ = \mathbb{Q}_+ \cup \{\infty\}$ avec l'opérateur standard d'addition étendu de manière à ce que $v+\infty=\infty$ pour tout $v \in \mathbb{Q}_+$.

De manière classique, les formules et circuits propositionnels sont définis sur un ensemble de variables booléennes $X=\{x_1,...,x_p\}$, les constantes $\top$ (vrai) et $\bot$ (faux), et les connecteurs $\neg$ (négation), $\wedge$ (conjonction) et $\vee$ (disjonction). Un littéral est une variable x_i ou sa négation $\neg x_i$, aussi noté $\overline{x}_i$. Un terme (resp. une clause) est une conjonction (resp. disjonction) de littéraux. Pour un sous-ensemble $Y \subseteq X$, une affectation partielle $\vec{y}$ sur Y est un vecteur de $2^{|Y|}$, qui peut être représenté de manière équivalente par un terme canonique sur Y, *i.e.* un terme cohérent, de $|Y|$ littéraux, chacun défini sur une variable de Y qui apparaît une unique fois dans le terme. On utilise aussi $\vec{Y}$ pour noter l'ensemble des affectations partielles sur Y, et $\vec{X}$ pour représenter l'ensemble $\{0,1\}^n$ de toutes les affectations totales, appelées *interprétations*.

Un réseau de contraintes valuées (VCN, pour *valued constraint network*) booléen P est composé d'un ensemble $X=\{x_1,...,x_p\}$ de variables booléennes et d'un ensemble $C=\{C_1,...,C_m\}$ de *contraintes valuées*. Une contrainte valuée $C_i=(Y_i,f_i)$ de C est définie par une *portée* $Y_i \subseteq X$ et une *fonction de coût* $f_i : \vec{Y}_i \to \overline{\mathbb{Q}}_+$. L'*arité* d'une contrainte valuée C_i est donnée par le cardinal $|Y_i|$ de sa portée. Si l'image de f_i est $\{0,\infty\}$, C_i est une *contrainte dure*; dans le cas contraire, il s'agit d'une *contrainte souple*, ou *potentiel*. Notons qu'un potentiel peut admettre ∞ parmi ses valeurs possibles, dans le but de permettre de représenter une exigence de l'utilisateur. Pour une contrainte valuée C_i et une affectation $\vec{y} \in \vec{Y}$, telle que $Y_i \subseteq Y$, on note $C_i(\vec{y})$ la valeur de $f_i(\vec{y}_i)$, où $\vec{y}_i$ est la projection de $\vec{y}$ sur Y_i. Le coût total d'une interprétation $\vec{x}$ dans P est donné par $P(\vec{x}) = \sum_{i=1}^{m} C_i(\vec{x})$. Une solution réalisable de P est une interprétation $\vec{x}$ telle que $P(\vec{x}) < \infty$, et une *solution minimale* de P est une solution réalisable de coût minimum. Le problème P est dit *faisable* si il admet au moins une solution réalisable; il est dit *infaisable* dans le cas contraire.

Nous utilisons fréquemment dans la suite deux opérations sur les contraintes valuées, le conditionnement et la restriction. Pour une fonction de coût $f_i : \vec{Y}_i \to \overline{\mathbb{Q}}_+$ et une affectation partielle $\vec{z} \in \vec{Z}$ (avec $Z \subseteq Y$), le *conditionnement* de f_i sur $\vec{z}$ est la fonction $f_i|\vec{z} : \vec{Y}_i \to \overline{\mathbb{Q}}_+$ où $(f_i|\vec{z})(\vec{y}_i) = f_i(\vec{y}_i)$ si $\vec{y}_i$ est cohérent avec $\vec{z}$, et $(f_i|\vec{z})(\vec{y}_i) = \infty$ dans le cas contraire. Étant donné un ensemble de contraintes $\mathcal{C}=\{C_1,...,C_m\}$ et un ensemble de variables $Y \subseteq X$, la *restriction* de $\mathcal{C}$ à Y, notée $\mathcal{C}_Y$, est le sous-ensemble de contraintes $\{C_i \in \mathcal{C} \mid Y_i \subseteq Y\}$.

La motivation principale de ce travail est l'analyse de la complexité des problèmes de minimisation dans lesquels l'ensemble de contraintes dures a été compilé en une unique contrainte. On suppose que pour cette contrainte il existe un algorithme en temps polynomial pour décider s'il existe ou non une extension à toute affectation partielle donnée qui soit une solution réalisable. Plus précisément, on étudie la complexité pour les réseaux de contraintes valuées composés d'une unique contrainte dure définie sur un langage propositionnel $\mathcal{L}$, et d'un ensemble de contraintes souples définies sur une famille de contraintes $\mathcal{F}$. Dans ce formalisme, tout VCN peut être représenté par un triplet (X,C,f) où C est la contrainte dure et f est la fonction de coût, représentée par un ensemble $\{C_i\}_{i=1}^m$ de contraintes souples.

Définition 1. ***MIN***$[\mathcal{L},\mathcal{F}]$ *est le problème d'optimisation suivant :*

- **Entrée :** *un réseau de contraintes valuées* $P=(X,C,f)$*, où* $C \in \mathcal{L}$ *et* $f \in \mathcal{F}$*;*
- **Sortie :** $\operatorname{argmin}_{\vec{x} \in \vec{X}}(C(\vec{x})+f(\vec{x}))$, i.e. *une solution minimale de* P *si* P *est faisable, et* $\bot$ *sinon.*

3 Représentation des contraintes dures

Les langages $\mathcal{L}$ examinés dans cet article sont des langages *complets* pour la logique propositionnelle : toute contrainte dure C peut être représentée par un circuit booléen dans $\mathcal{L}$. Dans la littérature de la compilation de connaissances, ces langages sont classés en fonction de leurs concision et le fait qu'ils possèdent ou non un algorithme en temps polynomial pour certaines requêtes et transformations [11]. On dit qu'un langage $\mathcal{L}1$ est *au moins aussi succinct* (concis) qu'un langage $\mathcal{L}2$, noté $\mathcal{L}1 \leq_s \mathcal{L}2$ si il existe un polynôme p tel que tout circuit C_2 de $\mathcal{L}2$ a un équivalent C_1 dans $\mathcal{L}1$ qui satisfait $|C_1| \leq p(|C_2|)$, où la taille $|C|$ d'une contrainte C exprimée par un circuit booléen est donnée par le nombre de ses arcs. On dit aussi que

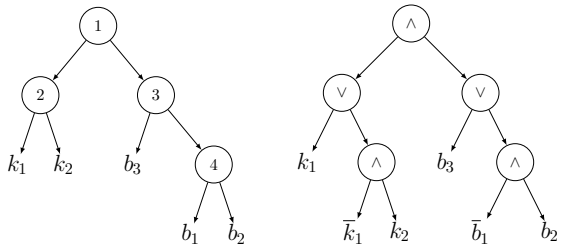


FIGURE 1 – Un vtree T (gauche) et un circuit $\texttt{DNNF}_T$ (droite).

$\mathcal{L}1$ est *(strictement) plus succinct* que $\mathcal{L}2$, noté $\mathcal{L}1 <_s \mathcal{L}2$ si $\mathcal{L}1$ est au moins aussi succinct que $\mathcal{L}2$ mais que l'inverse n'est pas vrai. Les requêtes sont utilisées pour extraire de l'information d'un circuit sans le modifier, et le but des transformations est de générer un nouveau circuit à partir d'un ou plusieurs circuits existants. Tous les langages auxquels nous nous intéressons ici satisfont la requête **CO**, qui vérifie la cohérence (*i.e.* l'existence d'une solution réalisable) de tout circuit C de $\mathcal{L}$, et la transformation **CD**, qui associe à tout circuit $C \in \mathcal{L}$ et toute affectation partielle $\vec{y}$ un circuit de $\mathcal{L}$ équivalent à $C|\vec{y}$, le conditionnement de C sur $\vec{y}$. En combinant ces deux propriétés, on montre immédiatement que $\mathcal{L}$ possède un algorithme en temps polynomial pour décider si une affectation partielle peut être étendue ou non à une solution réalisable.

Rappelons que NNF est l'ensemble des graphes dirigés acycliques possédant une racine où chaque nœud feuille est étiqueté par $\bot$, $\top$, ou un littéral sur X, et où chaque nœud interne est étiqueté soit par $\wedge$, soit par $\vee$. Pour un nœud N_i et une contrainte C, on note $Var(N_i)$ l'ensemble des variables qui étiquettent les nœuds feuilles accessibles depuis N_i. On note par ailleurs $Sol(N_i)$ l'ensemble des affectations $\vec{y}_i \in \vec{Y}_i$ telles que $C_i(\vec{y}_i) \neq \infty$, où $Y_i = Var(N_i)$ et C_i est le circuit NNF dont la racine est N_i. Par extension, on écrit $Var(C)$ (resp. $Sol(C)$) comme une abréviation de $Var(N)$ (resp. $Sol(N)$), où N est la racine de C.

Le langage NNF peut être raffiné en ajoutant des conditions sur les nœuds des circuits. Par exemple, le sous-langage des formules NNF *décomposables* [8] est défini de la manière suivante :

Définition 2 (DNNF). *Un circuit NNF C est dit* décomposable *si pour tout nœud ET N dans C ayant pour fils $N_1,...,N_q$, on a $Var(N_i) \cap Var(N_j) = \emptyset$ pour tout $i,j \in [q]$ avec $i \neq j$. L'ensemble des formules NNF décomposables est noté DNNF.*

Le langage DNNF peut être à son tour être raffiné en imposant des restrictions structurelles sur les nœuds en utilisant la notion de vtree introduite dans [26]. Formellement, un *vtree* est un arbre binaire T pour lequel les feuilles sont en correspondance une à une avec les variables de X. Pour un nœud t dans un vtree T, on note $Var(t)$ l'ensemble des feuilles dans le sous-arbre de T dont la racine est t. On note respectivement t_l et t_r les fils de gauche et de droite du nœud t. Un nœud interne t d'un vtree est appelé un *nœud de Shannon* si son fils gauche est une feuille, et un *nœud de décomposition* sinon ; on dit d'un vtree T dont tous les nœuds internes sont des nœuds de Shannon qu'il est *linéaire à droite*.

On dit qu'un nœud interne N_i d'un circuit NNF respecte un vtree T si il existe un nœud t dans T tel que, pour tout fils N_i de N, on a $Var(N_i) \subseteq Var(t_l)$ ou $Var(N_i) \subseteq Var(t_r)$.

Définition 3 ($\texttt{DNNF}_T$ et SDNNF). *Étant donné un vtree T, $\texttt{DNNF}_T$ est l'ensemble des circuits DNNF pour lesquels tous les nœuds ET respectent T. La classe SDNNF des circuits DNNF* structurés *est l'union des langages $\texttt{DNNF}_T$ définis sur tous les vtrees T.*

L'ensemble des circuits SDNNF considérés dans cette étude sont définis sur des nœuds ET binaires, ce qui n'est pas une restriction importante dans la mesure où toute formule C de SDNNF peut être transformée en une SDNNF équivalente de taille linéaire en $|C|$ et dont tous les nœuds de conjonction admettent exactement deux fils. Par analogie avec la terminologie employée pour les vtrees, un nœud ET N_i de C est un *nœud de Shannon* si au plus un de ses fils est un nœud interne ; sinon on le désigne comme *nœud de décomposition*. Par exemple, le circuit $\texttt{DNNF}_T$ donné à la figure 1 possède un nœud de décomposition (sa racine), et deux nœuds de Shannon : $\overline{k}_1 \wedge k_2$, et $\overline{b}_1 \wedge b_2$.

On trouve dans la littérature deux sous-classes de SDNNF bien connues : le langage SDD des *sentential decision diagrams* [10] et le langage OBDD des *diagrammes de décision ordonnés* [3]. On désigne par le terme *boîte* un nœud ET admettant exactement deux nœuds fils p et s, respectivement nommés *prime* et *sub*, qui sont étiquetés par une constante, un littéral, ou un nœud OU. Un nœud OU dont les fils sont $N_1,...,N_m$ est un *nœud partition* si il existe une partition $\{Y,Z\}$ de $Var(N)$ telle que :

- chaque nœud N_i est une boîte $p_i \wedge s_i$ avec $Var(p_i) = Y$ et $Var(s_i) \subseteq Z$;
- les *primes* de toute paire de fils sont mutuellement exclusifs, *i.e.* $Sol(p_i) \cap Sol(p_j) = \emptyset$ pour tout $i,j \in [m]$ avec $i \neq j$;
- la disjonction de l'ensemble des *primes* est valide, *i.e.* $Sol(p_1) \cup \cdots \cup Sol(p_m) = \vec{Y}$.

Définition 4 ($\texttt{SDD}_T$ et SDD). *Pour un vtree T, $\texttt{SDD}_T$ est le langage des circuits $\texttt{DNNF}_T$ dont la racine est un nœud OU tel que tout nœud ET est une boîte respectant T, et tout nœud OU est un nœud partition. SDD est l'union des langages $\texttt{SDD}_T$ définis sur tous les vtrees T.*

Exemple 1. *Un exemple de circuit $\texttt{SDD}_T$ C, dont le vtree T est celui de la figure 1, est donné à la figure 2. C*

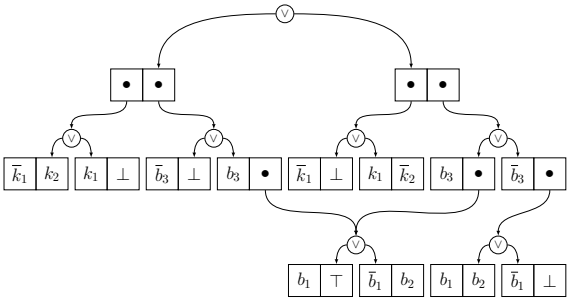


FIGURE 2 – Un circuit $\texttt{SDD}_T$ représentant les contraintes d'un problème de configuration de maison.

représente les contraintes d'un problème de configuration de maisons, où k_1 et k_2 représentent les types de cuisine (respectivement standard ou premium), et b_1, b_2 et b_3 sont les différentes alternatives pour les types de chambre (petite, moyenne, et grande). Les maisons réalisables sont celles ayant une unique cuisine et au moins deux chambres ; de plus, seules les maisons avec au moins une grande chambre peuvent avoir une cuisine premium. En termes plus formels, le circuit C encode la conjonction des contraintes $k_1+k_2=1$, $b_1+b_2+b_3\geq 2$ et $\overline{k}_2\vee b_3$.

En se basant sur ces notations, on peut voir un circuit $\texttt{OBDD}_T$ [2] comme un *sentential decison diagram* défini à partir d'un vtree linéaire à droite [10].

Définition 5 (OBDD). *OBDD est l'union des langages $\texttt{SDD}_T$ définis sur l'ensemble des vtrees T linéaires à droite.*

L'ensemble des langages susmentionnés peuvent être restreints en ajoutant des conditions sur les *arcs* de leurs circuits.

Définition 6 (acy-NNF). *Un circuit NNF est* fortement acyclique *si le graphe non dirigé qui lui est associé est acyclique. On note* acy-NNF *de l'ensemble des circuits NNF fortement acycliques.*

Par extension, $\texttt{acy-DNNF}_T$ est défini par $\texttt{acy-NNF}\cap\texttt{DNNF}_T$, et acy-SDNNF est l'union des langages $\texttt{acy-DNNF}_T$ définis sur l'ensemble des vtrees T. Puisque le langage DNF des formules sous forme normale disjonctive est complet et que chacune de ses formules peut être représentée en temps linéaire comme une formule $\texttt{DNNF}_T$ fortement acyclique pour n'importe quel vtree T, il s'ensuit que $\texttt{acy-DNNF}_T$ et acy-SDNNF sont des langages complets pour la logique propositionnelle.

À la lumière des définitions ci-avant, on peut vérifier que SDD et acy-SDNNF sont des sous-ensembles de SDNNF. Bien qu'on pourrait penser que SDD soit strictement plus succinct que acy-SDNNF, dans la mesure où les circuits SDD sont des graphes dirigés acycliques et les circuits acy-SDNNF sont des arbres, il n'en est rien, à moins que la hiérarchie polynomiale ne s'effondre. En effet, d'un côté, on sait dans [10] que $\texttt{d-DNNF}\leq_s\texttt{SDD}$, où d-DNNF est l'ensemble des circuits DNNF déterministes. D'un autre côté, on sait aussi que $\texttt{acy-DNNF}_T\leq_s\texttt{DNF}$, puisque nous avons vu précédemment que toute formule DNF peut être réécrite en temps polynomial comme un circuit $\texttt{DNNF}_T$ fortement acyclique, peu importe T. En se basant sur ces deux inégalités, supposer $\texttt{SDD}\leq_s\texttt{acy-SDNNF}$ permettrait de dériver $\texttt{d-DNNF}\leq_s\texttt{DNF}$, ce qui est impossible à moins que la hiérarchie polynomiale ne s'effondre [11].

De ce fait, du point de vue de la concision, les langages SDD et acy-SDNNF sont des candidats indiqués pour la compilation de contraintes dures. Comme SDD est un sur-ensemble de OBDD, SDD peut être employé, par exemple, pour encoder en temps polynomial des contraintes de cardinalité [14]. De plus, acy-SDNNF est un fragment notable de DNNF qui permet de représenter tout circuit SDNNF de profondeur *bornée* (en le dépliant), et les algorithmes de compilation procédant de la racine aux feuilles [9] peuvent être adaptés pour générer directement des circuits $\texttt{acy-DNNF}_T$ en désactivant le cache du compilateur. On peut aussi employer des compilateurs procédant des feuilles vers la racine pour construire de telles représentations dans la mesure où $\texttt{acy-DNNF}_T$ satisfait les transformations $\vee\mathbf{C}$ et $\wedge\mathbf{BC}$ [11].

4 Représentation des fonctions de coût

Reprenant la terminologie de [4], un *langage de contraintes valuées* est un ensemble $\mathcal{F}$ de fonctions de coût à valeurs dans $\overline{\mathbb{Q}}_+$ d'arités quelconques. Dans cette étude, nous utilisons le terme *famille de contraintes valuées* pour faire référence à $\mathcal{F}$, afin d'éviter toute confusion avec la notion de *langage de contrainte* $\mathcal{L}$ utilisée pour la description des contraintes dures.

Nous nous focalisons sur des familles de contraintes valuées $\mathcal{F}$ closes pour l'addition et le conditionnement :

- si f et g sont deux fonctions de $\mathcal{F}$, alors $f+g\in\mathcal{F}$;
- si $f:\vec{X}\to\overline{\mathbb{Q}}_+$ est une fonction de $\mathcal{F}$, et $\vec{z}\in\vec{Z}$ est une affectation partielle sur $Z\subseteq X$, alors $f|\vec{z}\in\mathcal{F}$.

De nombreuses familles de contraintes respectent ces hypothèses, comme les fonctions polynomiales. Soit $\texttt{POLY}_k$ l'ensemble des fonctions $f:\{0,1\}^j\to\overline{\mathbb{Q}}_+$ d'arité $j\in[k]$, et soit POLY l'union de l'ensemble des langages $\texttt{POLY}_k$ pour $k\in\mathbb{N}$. Tout sous-ensemble $\mathcal{F}$ de POLY est nommé *famille de contraintes polynomiales*, ce qui inclut en particulier les familles $\texttt{POLY}_1$ et $\texttt{POLY}_2$ correspondant respectivement aux familles *linéaires* et *quadratiques*. Les contraintes souples dont la fonction de coût associée est dans $\texttt{POLY}_k$ peuvent être décrites de manière classique par des polynômes de degré k, c'est-à-dire par des sommes pondérées de termes canoniques. Par exemple, si $Y_i=\{x_1,x_2\}$ et f_i est donnée

2. $\texttt{OBDD}_T$ est habituellement noté $\texttt{OBDD}_<$; ce changement de notation ne pose cependant pas de problème dans la mesure où un vtree linéaire à droite peut être vu comme un ordre total sur les variables.

par la table $\{(00,1),(01,2),(10,3),(11,0)\}$, la contrainte binaire valuée $C_i = (Y_i, f_i)$ peut être représentée par le polynôme pondéré $(\overline{x}_1 \wedge \overline{x}_2) + 2(\overline{x}_1 \wedge x_2) + 3(x_1 \wedge \overline{x}_2)$.

Comme nous l'avons noté dans l'introduction, les fonctions de coût sous-modulaires occupent une place importante dans l'optimisation non linéaire. Formellement, une fonction $f : \vec{X} \to \overline{\mathbb{Q}}_+$ est sous-modulaire si pour tout $\vec{x}, \vec{y} \in \vec{X}$, on a $f(\vec{x} \wedge \vec{y}) + f(\vec{x} \vee \vec{y}) \leq f(\vec{x}) + f(\vec{y})$, où $\wedge$ et $\vee$ sont respectivement les opérateurs bit à bit ET et OU sur les booléens. Il est facile de constater que la sous-modularité est préservée par le conditionnement, et que, bien que toute fonction linéaire soit sous-modulaire, l'inverse n'est pas vrai. En effet, la disjonction pondérée $f(\vec{x}) = w(x_1 \vee \cdots \vee x_k)$, où $w \in \mathbb{Q}_+$, est une fonction sous-modulaire qui ne peut pas être exprimée à l'aide d'une fonction linéaire. De manière similaire, la fonction de budget $f(\vec{x}) = \min(r, w_1 x_1 + \cdots + w_k x_k)$, où $r \in \mathbb{Q}_+$ et $\vec{w} \in \mathbb{Q}_+^k$, est sous-modulaire mais pas linéaire. Soit $\texttt{SUB}_k$ le sous-ensemble de $\texttt{POLY}_k$ formé par l'ensemble des fonctions sous-modulaires d'arité au plus k, et soit $\texttt{SUB}$ l'union de l'ensemble des $\texttt{SUB}_k$ pour $k \in \mathbb{N}$, tout sous-ensemble $\mathcal{F}$ de $\texttt{SUB}$ est nommé *famille de contraintes sous-modulaires*. Comme auparavant, les contraintes souples définies sur $\texttt{SUB}_k$ peuvent être représentées par des polynômes. Pour les potentiels $C_i = (Y_i, f_i)$ définis sur la classe générale $\texttt{SUB}$, f_i est usuellement obtenue par un *oracle de valeurs*, c'est-à-dire un algorithme s'exécutant en temps polynomial qui associe à toute entrée $\vec{y}_i \in \vec{Y}_i$ la valeur $f(\vec{y}_i)$.

Bien qu'il soit connu que toute fonction de coût polynomiale peut être exprimée par des sommes de potentiels de $\texttt{POLY}_2$ [2], une fonction sous-modulaire quelconque ne peut pas nécessairement s'exprimer à partir de $\texttt{SUB}_2$ [36]. Cela n'enlève en rien l'intérêt de la famille $\texttt{SUB}_2$, puisque le fait que les fonctions quadratiques sous-modulaires puissent être encodées par des *cuts* dans un graphe dirigé implique que la minimisation non contrainte dans $\texttt{SUB}_2$ peut être réalisee en temps $\mathcal{O}(n^3)$ (où n est la taille de l'entrée) ; alors que le meilleur algorithme connu pour $\texttt{SUB}$ nécessite un temps en $\mathcal{O}(n^5 VO + n^6)$, où VO est le temps nécessaire à l'exécution de l'oracle de valeurs [21].

Étant donné une famille de contraintes valuées $\mathcal{F}$ et une fonction de coût f représentée par un ensemble de contraintes souples $\{(Y_i, f_i)\}_{i=1}^m$, on dit que f *appartient à* $\mathcal{F}$, noté $f \in \mathcal{F}$ si $f_i \in \mathcal{F}$ pour tout $i \in [m]$. La *taille* de f est définie par $|f| = \sum_{i=1}^m |f_i|$. Pour les familles polynomiales $\mathcal{F} \subseteq \texttt{POLY}_k$, la taille $|f_i|$ de chaque fonction de coût f_i est donnée par le nombre de termes canoniques dans sa représentation par un polynôme.

On associe à chaque fonction de coût f donnée par un ensemble de potentiels $\{(Y_i, f_i)\}_{i=1}^m$ un hypergraphe $\mathcal{H}_f$ qui capture la structure de f, dont l'ensemble des sommets est défini par $Var(f) = \bigcup_{i=1}^m Y_i$, et l'ensemble des hyperarêtes est donné par $\{Y_i\}_{i=1}^m$. La taille $|\mathcal{H}_f|$ de $\mathcal{H}_f$ est définie comme la somme de la taille de ses hyperarêtes, à savoir $|H_f| = \sum_{i=1}^m |Y_i|$.

De manière générale, une famille de contraintes $\mathcal{F}$ impose peu de restrictions sur la structure d'une fonction de coût $f \in \mathcal{F}$. Par exemple, si $\mathcal{F}$ est la classe $\texttt{SUB}_2$, $\mathcal{H}_f$ peut *a priori* être n'importe quel sous-graphe du graphe complet sur $Var(f)$. Afin de mettre en lumière les relations structurelles entre la fonction de coût f et la contrainte dure C, nous avons besoin de définitions permettant de quantifier la similarité (ou la dissimilarité) entre les structures de f et de C.

Définition 7 (Compatibilité structurelle)**.** *Soit C un circuit* SDNNF *avec $Var(C) \subseteq X$, et soit Y un sous-ensemble de X. Alors,*

- *Y est* compatible avec un nœud de décomposition (ET) *$N = N_l \wedge N_r$ de C si $Y \cap Var(N) \neq \emptyset$ implique soit $Y \subseteq Var(N_l)$, soit $Y \subseteq Var(N_r)$, mais pas les deux à la fois.*
- *Y est* compatible avec un nœud OU *N de C si $Y \cap Var(N) \neq \emptyset$ implique $Y \subseteq Var(N)$.*

On dit que Y est faiblement *compatible avec C, noté $Y \sim C$, si Y est compatible avec chaque nœud de décomposition de C. Y est dit* fortement *compatible avec C, noté $Y \sim^* C$, si Y est compatible avec chaque nœud de décomposition et chaque nœud OU de C. Par extension, une fonction de coût f est* faiblement *(resp.* fortement*)* compatible *avec C si $Y_i \sim C$ (resp. $Y_i \sim^* C$) pour tout $Y_i \in \mathcal{H}_f$.*

Intuitivement, la propriété de faible compatibilité indique que si la portée Y_i d'un potentiel (Y_i, f_i) partage des variables avec un nœud de décomposition $N = N_l \wedge N_r$, alors Y_i doit être couvert par les variables *d'exactement un* des fils de N. La forte compatibilité ajoute que si Y_i partage des variables avec un nœud OU N, alors Y_i doit être couvert par *toutes* les variables de N. Notons que si Y_i est un ensemble singleton, il est nécessairement fortement compatible avec n'importe quel nœud de C. Notons aussi que ces propriétés de compatibilité n'impliquent aucune restriction sur les nœuds de Shannon ET de C ; la mesure de dissimilarité présentée plus bas est de ce fait indépendante du nombre de nœuds de Shannon présents dans la contrainte dure.

Exemple 2. *On considère à nouveau le problème de configuration de l'exemple 1 et la fonction de coût $f(\vec{k}, \vec{b}) = p(k_1 \vee k_2) + \min(r, q_1 b_1 + q_2 b_2 + q_3 b_3)$, tel que $p, r \in \mathbb{Q}_+$ et $q \in \mathbb{Q}_+^3$ sont des coûts. Le premier terme de f est une disjonction indiquant qu'un même coût p est associé à tout ensemble non vide de cuisines, quand le second terme de f est un potentiel de budget indiquant qu'une pénalité maximale de r sera appliqué en ce qui concerne le prix des chambres. Selon les définitions précédentes, f est une fonction sous-modulaire cubique, et elle est fortement compatible avec la contrainte* $\texttt{acy-DNNF}_T$ *de la figure 1. Cependant, f n'est pas fortement compatible avec la contrainte* SDD *de la figure 2, puisque la portée*

$\{b_1, b_2, b_3\}$ *associée au second terme de* f *n'est pas compatible avec, par exemple, le nœud* $b_1 \vee (b_1 \wedge b_2)$. *En revanche, la fonction de coût sous-modulaire quadratique* $g(\vec{k},\vec{b}) = p(k_1 \vee k_2) + \min(r, q_1 b_1 + q_2 b_2) + q_3 b_3$ *est fortement compatible avec cette contrainte* SDD.

Définition 8 (Dissimilarité structurelle)**.** *Soit* C *une contrainte* SDNNF *avec* $Var(C) \subseteq X$, *et soit* Y *un sous-ensemble de* X. *Alors,*

— *la* faible *(resp.* forte*)* dissimilarité *entre* Y *et* C, *notée* $\delta(Y,C)$ *(resp.* $\delta^*(Y,C)$*), est le nombre minimal de variables qui peuvent être retirées de* Y *de manière à atteindre un sous-ensemble faiblement (resp. fortement) compatible avec* C :

$$\delta(Y,C) = \min_{Z \subseteq Y | Y \setminus Z \sim C} |Z|,$$
$$\delta^*(Y,C) = \min_{Z \subseteq Y | Y \setminus Z \sim^* C} |Z|.$$

— *Par extension, la* faible *(resp.* forte*)* dissimilarité *entre une fonction de coût* f *et* C *est la somme des faibles (resp. fortes) dissimilarités entre chacune de ses portées et* C :

$$\delta(f,C) = \sum_{Y \in \mathcal{H}_f} \delta(Y,C),$$
$$\delta^*(f,C) = \sum_{Y \in \mathcal{H}_f} \delta^*(Y,C).$$

Par exemple, la forte dissimilarité entre la fonction de coût $f(\vec{k},\vec{b}) = p(k_1 \vee k_2) + \min(r, q_1 b_1 + q_2 b_2 + q_3 b_3)$ et la contrainte $C \in$ SDNNF de la figure 1 vaut 1.

Proposition 1. *Soit* C *une contrainte* SDNNF *avec* $Var(C) \subseteq X$, *et soit* f *une fonction de coût avec* $Var(f) \subseteq X$. $\delta(f,C)$ *et* $\delta^*(f,C)$ *peuvent être tous deux calculés en temps* $\mathcal{O}(|C||H_f|)$.

La preuve de cette proposition, de même que les preuves des propositions suivantes, est diponible en ligne à l'adresse `http://www.cril.univ-artois.fr/~lonca/dnnf-fpt.pdf`.

5 Résultats de complexité

Comme nous l'avons mentionné dans l'introduction, le problème de minimisation sous contraintes **MIN**[DNNF,POLY_1] peut être résolu en temps polynomial [12]. Nos résultats de complexité peuvent être synthétisés par trois théorèmes clés qui établissent la traitabilité à paramètre fixé pour la requête de minimisation **MIN**$[\mathcal{L},\mathcal{F}]$ pour divers langages $\mathcal{L} \subseteq$ DNNF et familles de fonctions de coût $\mathcal{F}$ non linéaires. Dans la suite, nous considérons que les fonctions de coût f d'entrée sont définies sur la totalité de l'ensemble des variables X, ce qui n'est pas une grande restriction puisque toute fonction f telle que $Var(f) \not\subseteq X$ peut être étendue pour porter sur X en lui ajoutant des *potentiels nuls* de la forme $(\{x_i\},0)$ pour tout $x_i \in X \setminus Var(f)$, où 0 est la fonction constante qui renvoie zéro.

5.1 Minimisation quadratique sous DNNF

Nous débutons par le problème de la minimisation de fonctions de coût quadratiques sous contrainte DNNF_T. Rappelons que le problème de minimisation d'une somme quelconque de potentiels quadratiques avec $C = \top$ est NP-difficile [33]. Le résultat suivant montre que la minimisation quadratique sous contrainte DNNF est traitable à paramètre fixé k où k est le nombre de contraintes binaires (valuées) de la fonction de coût.

Théorème 1. **MIN**[DNNF,POLY_2] *est* FPT *à paramètre* k *où* k *est le nombre de portées binaires de* $\mathcal{H}_f$.

5.2 Minimisation sous-modulaire sous acy-SDNNF

Nous nous intéressons maintenant aux fonctions de coût sous-modulaires, et débutons avec un résultat négatif indiquant que même dans le cas des DNNF (structurées) fortement acycliques, le problème de minimisation est difficile.

Proposition 2. **MIN**[acy-SDNNF,SUB] *est* NP*-difficile.*

Nous montrons maintenant que si la contrainte C est dans acy-SDNNF, et si la fonction de coût sous-modulaire f est faiblement compatible avec C, alors la minimisation de f sous C est dans P. Afin de prouver ce résultat , nous utilisons un algorithme de minimisation procédant de la racine vers les feuilles (TDM), qui décompose de manière itérative la fonction de coût f entre les nœuds de la contrainte acy-SDNNF C. On rappelle que $f|_\ell$ est le conditionnement de f par le littéral (ou affectation partielle de taille 1) ℓ, et f_Y est la restriction de f à l'ensemble de variables Y.

Exemple 3. *Afin d'illustrer le comportement de l'algorithme TDM, considérons la contrainte* DNNF_T *fortement acyclique* C *de la figure 1 avec la fonction* $f(\vec{k},\vec{b}) = p(k_1 \vee k_2) + \min(r, q_1 b_1 + q_2 b_2 + q_3 b_3)$, *où* $q_3 < r < q_1 < q_2$. *Comme mentionné dans l'exemple 2, on sait que* f *est faiblement compatible avec* C. *Puisque la racine* N *de* C *est un nœud de décomposition ET (ligne 7), la procédure appelle récursivement* TDM *sur* $f_l(\vec{k}) = p(k_1 \vee k_2)$ *pour le fils gauche* $N_l = k_1 \vee (\overline{k}_1 \wedge k_2)$, *et* $f_r(\vec{b}) = \min(r, q_1 b_1 + q_2 b_2 + q_3 b_3)$ *pour le fils droit* $N_r = b_3 \vee (\overline{b}_1 \wedge b_2)$. *Pour le fils gauche* N_l, *qui est un nœud OU (ligne 11), la procédure appelle récursivement* TDM *sur* $f_l(\vec{k})$ *sous contrainte* k_1, *et* $f_l(\vec{k})$ *sous contrainte* $(\overline{k}_1 \wedge k_2)$. *À la ligne 3, les termes minimisant* $(f_l(\vec{k}))_{|k_1}$ *sont ceux de l'ensemble* $\{k_1 \wedge k_2, k_1 \wedge \overline{k}_2\}$ *avec le coût* p. *À la ligne 7, le terme minimisant* $f_l(\vec{k})$ *tel que* $(\overline{k}_1 \wedge k_2)$ *soit vrai est* $\overline{k}_1 k_2$ *avec le coût* p. *Pour résumer, l'affectation partielle retournée par TDM pour* $f_l(\vec{k})$ *sur* N_l *est un terme dans l'ensemble* $\{k_1 k_2, k_1 \overline{k}_2, \overline{k}_1 k_2\}$. *De la même manière, le terme retourné par TDM pour* $f_r(\vec{b})$ *sous la contrainte du fils droit* N_r *est* $\overline{b}_1 \overline{b}_2 b_3$ *avec le coût* q_3.

Algorithme 1 : TDM : *Top Down Minimization*

Entrées : Une fonction de coût sous-modulaire f et un circuit SDNNF C dont la racine est N
Sorties : Une affectation $\vec{x} \in \vec{X}$ qui minimise f si C est cohérent, ou $\bot$ sinon

si $N = \bot$ **alors retourner** $\bot$
si $N = \top$ **alors retourner** $\text{argmin}_{\vec{y} \in \vec{Y}} f(\vec{y})$, où $Y = Var(f)$
si $N = \ell$ **alors**
  retourner $\text{argmin}_{\vec{y} \in \vec{Y}} f|_\ell(\vec{y})$, où $Y = Var(f)$
si $N = \ell \wedge N'$ est un nœud de Shannon **alors**
  retourner $\text{TDM}(f|_\ell, N')$
si $N = N_1 \wedge N_2$ est un nœud décomposition **alors**
  $\vec{y}_0 \leftarrow \text{TDM}(f_{Var(f) \setminus Var(N)}, \top)$
  $\vec{y}_i \leftarrow \text{TDM}(f_{Var(N_i)}, N_i)$ pour tout $i \in [2]$
  retourner $\vec{y}_0 \wedge \vec{y}_1 \wedge \vec{y}_2$
si $N = N_1 \vee ... \vee N_q$ **alors**
  $\vec{y}_i \leftarrow \text{TDM}(f, N_i)$ pour tout $i \in [q]$
  retourner $\text{argmin}\{f(\vec{y}_i)\}$

Algorithme 2 : BUM : *Bottom Up Minimization*

Entrées : Une fonction de coût sous-modulaire f et un circuit SDD C dont la racine est N
Sorties : Une affectation $\vec{x} \in \vec{X}$ qui minimise f si C est cohérent, et $\bot$ sinon

pour chaque *nœud N de C dans l'ordre topologique inverse* **faire**
  si $N = \bot$ **alors** $f_N \leftarrow \{(\emptyset, \infty)\}$
  si $N = \top$ **alors** $f_N \leftarrow \emptyset$
  si $N = \ell$ **alors**
    $f_N \leftarrow \{(Y_i, f_i|_\ell) \mid (Y_i, f_i) \in f \text{ et } Y \cap Var(\ell) \neq \emptyset\}$
  si $N = N_1 \wedge N_2$ **alors** $f_N \leftarrow f_{N_1} \cup f_{N_2}$
  si $N = N_1 \vee ... \vee N_q$ **alors**
    $\vec{y}_i \leftarrow \text{argmin}_{\vec{y} \in \vec{Y}} f_{N_i}(\vec{y})$ pour tout $i \in [q]$
    $f_N \leftarrow (Y, \{(\vec{y}, f(\vec{y}))\})$, où $\vec{y} = \text{argmin}_{i=1}^{q} f(\vec{y}_i)$
retourner $\text{argmin}_{\vec{x} \in \vec{X}} f_N(\vec{x}) + f_\top(\vec{x})$

Proposition 3. **MIN**[acy-SDNNF, SUB] *est dans* P *si la fonction de coût f est faiblement compatible avec la contrainte dure C.*

Corollaire 1. **MIN**[DNF,SUB] *est dans* P.

Pour résumer, on sait que la minimisation sous-modulaire sous contrainte acy-SDNNF est NP-difficile dans le cas général, mais traitable si la fonction de coût est faiblement compatible avec la contrainte dure. Cela nous permet maintenant d'opposer une restriction au résultat général de non-traitabilité de la proposition 2, utilisant pour cela la notion de faible dissimilarité.

Théorème 2. **MIN**[acy-SDNNF, SUB] *est* FPT *à paramètre δ.*

Corollaire 2. **MIN**[SDNNF, SUB] *est* FPT *à paramètre $d+\delta$, où d est la profondeur de la contrainte SDNNF.*

5.3 Optimisation sous-modulaire sous SDD

La dernière partie de cette étude concerne la minimisation sous-modulaire sous contrainte SDD. Encore une fois, nous débutons par un résultat négatif fort indiquant que la minimisation sous-modulaire quadratique sous contrainte SDD est NP-difficile, même lorsque la contrainte dure est un circuit OBDD.

Proposition 4. **MIN**[OBDD,SUB_2] *est* NP*-difficile.*

Cependant, on peut montrer que la minimisation sous-modulaire sous contrainte SDD est dans P, dans le cas où la fonction de coût est fortement compatible avec la contrainte. Ce résultat est établi en employant un algorithme de minimisation procédant des feuilles à la racine (BUM), qui calcule dans un premier temps un ordre topologique inverse de la contrainte d'entrée SDD C, puis qui simplifie et collecte de manière itérative les potentiels de f dont la portée est couverte par le nœud N courant. On utilise ici f_N pour abréger $f_{Var(N)}$, et on note $(\emptyset, \infty)$ le potentiel irréalisable, où ∞ est vu comme une fonction constante. Puisque $Var(C)$ peut être un sous-ensemble strict de X, f (qui est par hypothèse définie sur $Var(f) = X$) peut inclure des variables absentes de C. De ce fait, une dernière étape de minimisation est appliquée sur la sous-fonction non-contrainte $f_\top = f_{Var(f) \setminus Var(N)}$, où N est la racine de C.

Proposition 5. **MIN**[SDD,SUB] *est dans* P *si la fonction de coût en entrée f est fortement compatible avec la contrainte dure C.*

Exemple 4. *Considérons le circuit SDD de la Figure 2 et la fonction de coût $g(\vec{k}, \vec{b}) = p(k_1 \vee k_2) + \min(r, q_1 b_1 + q_2 b_2) + q_3 b_3$, avec $q_3 < q_2 < r < q_1$ et $r < q_2 + q_3$, pour illustrer l'algorithme BUM. La fonction associée à la boîte $b_1 \wedge \top$ à la dernière* ligne *de la figure est composée des deux potentiels associés à ses fils b_1 et $\top$, c'est-à-dire respectivement $(\{b_1, b_2\}, \min(r, q_1 b_1 + q_2 b_2) \mid q_1)$ et $\varnothing$. En ce qui concerne la deuxième boîte $\neg b_1 \wedge b_2$ de la dernière ligne, la fonction associée est composée des deux potentiels $(\{b_1, b_2\}, \min(r, q_1 b_1 + q_2 b_2) \mid \bar{b}_1)$ et $(\{b_1, b_2\}, \min(r, q_1 b_1 + q_2 b_2) \mid b_2)$. De ce fait, la fonction g_N associée au nœud OU père des deux boîtes est construite à partir des affectations partielles des variables de $\{b_1, b_2\}$ qui minimisent $\min(\min(r, q_1 b_1 + q_2 b_2) \mid q_1, (\min(r, q_1 b_1 + q_2 b_2) \mid \bar{b}_1) + (\min(r, q_1 b_1 + q_2 b_2) \mid b_2)))$,*

	SDNNF	acy-SDNNF	SDD	OBDD	DNF
$\mathtt{POLY}_2$	k	k	k	k	k
SUB	$d{+}\delta$	δ	δ^*	δ^*	$-$

TABLE 1 – Paramètres de complexité utilisés dans les résultats de FPT. Ici, k est le nombre de portées binaires dans la fonction de coût, d est la profondeur de la contrainte dure, et $-$ indique que le problème est dans P.

c'est-à-dire $\bar{b}_1 \wedge b_2$ *(puisque* $q_2 < q_1$*). De ce fait, on a* $g_N = (\{b_1,b_2\},\min(r,q_1 b_1 + q_2 b_2) \mid \bar{b}_1 b_2)$*. L'application de cet algorithme des feuilles à la racine du circuit* SDD *permet de déterminer la valeur minimale* $p + r$ *pour l'affectation* $k_1 \wedge \bar{k}_2 \wedge b_1 \wedge b_2 \wedge \bar{b}_3$*.*

Théorème 3. **MIN**[SDD,SUB] *est* FPT *à paramètre* δ^**.*

6 DISCUSSION

Dans cet article, nous avons étudié la complexité de la minimisation des fonctions quadratiques et sous-modulaires sous contraintes DNNF, dont les résultats en terme de traitabilité à paramètre fixé sont rappelés à la table 1. D'un point de vue pratique, la minimisation sous-modulaire sous contrainte SDNNF (et ses sous-ensembles) peut être réalisée de manière efficace si la profondeur d de la contrainte C et la faible dissimilarité de la fonction d'entrée f (par rapport à C) sont relativement petites. D'un autre côté, la minimisation sous-modulaire sous contraintes SDD (et de ce fait, de OBDD) peut être effectuée efficacement si la forte dissimilarité entre f et C est faible, peu importe la profondeur du circuit SDD considéré. En ce qui concerne la minimisation sous-modulaire quadratique, la requête $\mathbf{MIN}[\mathtt{SDD}, \mathtt{SUB}_2]$ peut être résolue en temps $\mathcal{O}(|C|n^3 2^{\delta(f,C)})$ grâce à l'algorithme BUM.

Comparaison avec l'existant. De grands efforts ont été fournis dans le but d'identifier des familles de réseaux de contraintes valuées pour lesquelles l'optimisation est traitable. La plupart des travaux dans ce champ de recherche se sont basés sur trois approches principales, en fonction du type de restrictions proposées pour obtenir des classes traitables. La première consiste à identifier des propriété *structurelles* du VCN pour assurer la traitabilité ; on sait par exemple que le problème de minimisation est polynomial si la macro-structure du réseau à une (hyper)-*tree-width* bornée [19]. Dans un contexte similaire, différents langages de compilation ont été définis pour compiler la micro-structure d'un VCN en un circuit valué à partir duquel l'optimisation peut être réalisée en temps polynomial [34, 15, 22]. Il est important de noter que notre travail diffère de ceux-ci, qui compilent *à la fois* les contraintes dures et souples, alors que dans notre cas les potentiels ne sont connus qu'après la phase de compilation et peuvent varier en fonction de l'utilisateur. Les garanties de performances que nous recherchons excluant de pouvoir réaliser une lourde étape de compilation chaque fois qu'une nouvelle fonction de coût est considérée.

La seconde approche consiste à identifier des propriétés *algébriques* des contraintes valuées suffisamment restrictives pour assurer la traitabilité, peu importe les contraintes du réseau. Un classification complète de la complexité de l'optimisation pour les langages de contraintes valuées a été établie pour les VCN booléens [4], et montre que l'optimisation n'est facile que pour des fragments très restreints.

Nos travaux appartiennent à une troisième approche, *hybride*, qui se base sur des restrictions posées à la fois sur les structures et les fonctions. Les résultats négatifs forts obtenus ici pour la minimisation sous-modulaire sous contraintes indique que, de manière générale, les restrictions sur les structures et celles sur les langages ne peuvent pas être considérées séparément. En effet, même si les contraintes dures sont décrites par un matroïde pour lequel l'optimisation linéaire est dans P, et que la fonction de coût est sous-modulaire, le problème de minimisation correspondant est NP-difficile et généralement non approximable à un facteur près [17, 31]. Des classes traitables ont été obtenues par Cooper et Zivny [5, 6] en combinant de manière appropriée les restrictions sur la micro-structure du réseau et les restrictions sur les langages des fonctions de coût. Nos résultats exploitent de telles restrictions hybrides, mais couvrent un ensemble plus large de contraintes dures, les circuits DNNF.

Perspectives. À la lumière des présents résultats, une direction intéressante consisterait à considérer le problème de *maximisation* de fonctions sous-modulaires sous contrainte DNNF. Bien que la maximisation et la minimisation soient des problèmes équivalents pour les langages de contraintes valuées clos pour l'inverse $(-)$, comme le langage des fonctions de coût linéaires, ce n'est pas le cas en général pour les fonctions sous-modulaires. En particulier, le problème de la maximisation de fonctions sous-modulaires (monotones) est NP-difficile, mais approximable à un facteur près, dans le cas non contraint. Une question importante serait de déterminer si une telle borne est préservée sous contrainte DNNF.

Références

[1] F. Bacchus and A. Grove. Graphical models for preference and utility. In *Proc. of UAI '95*, pages 3–10, 1995.

[2] E. Boros and P. Hammer. Pseudo-Boolean Optimization. *Discrete Applied Mathematics*, 123(1) :155–225, 2002.

[3] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8) :677–691, 1986.

[4] D. A. Cohen, M. C. Cooper, P. Jeavons, and A. A. Krokhin. The complexity of soft constraint satisfaction. *Artif. Intell.*, 170(11) :983–1016, 2006.

[5] M. Cooper and S. Zivny. Hybrid tractability of valued constraint problems. *Artif. Intell.*, 175(9-10) :1555–1569, 2011.

[6] M. Cooper and S. Zivny. Tractable triangles and cross-free convexity in discrete optimisation. *J. Artif. Intell. Res. (JAIR)*, 44 :455–490, 2012.

[7] N. Creignou, S. Khanna, and M. Suda. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. Monographs on Discrete Mathematics and Applications. SIAM, 2001.

[8] A. Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4) :608–647, 2001.

[9] A. Darwiche. A compiler for deterministic, decomposable negation normal form. In *Proc. of AAAI*, pages 627–634, 2002.

[10] A. Darwiche. SDD : A new canonical representation of propositional knowledge bases. In *Proc. of IJCAI*, pages 819–826, 2011.

[11] A. Darwiche and P. Marquis. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)*, 17 :229–264, 2002.

[12] A. Darwiche and P. Marquis. Compiling propositional weighted bases. *Artif. Intell.*, 157(1-2) :81–113, 2004.

[13] G. Downey and R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

[14] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *JSAT*, 2(1-4) :1–26, 2006.

[15] H. Fargier and P. Marquis. On valued negation normal form formulas. In Manuela M. Veloso, editor, *Proc. of IJCAI*, pages 360–365, 2007.

[16] S. Fujishige. *Submodular Functions and Optimization*. Elsevier, 2005.

[17] G. Goel, C. Karande, P. Tripathi, and L. Wang. Approximability of combinatorial problems with multi-agent submodular cost functions. In *Proc. of FOCS*, pages 755–764, 2009.

[18] Ch. Gonzales and P. Perny. GAI networks for utility elicitation. In *Proc. of KR*, pages 224–234, 2004.

[19] G. Gottlob, G. Greco, and F. Scarcello. Tractable optimization problems through hypergraph-based structural restrictions. In *Proc. of ICALP*, pages 16–30, 2009.

[20] S. Iwata. Submodular function minimization. *Math. Program.*, 112(1) :45–64, 2008.

[21] S. Iwata and J. Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proc. of SODA*, pages 1230–1237, 2009.

[22] G. Katsirelos, N. Narodytska, and T. Walsh. The weighted grammar constraint. *Annals OR*, 184(1) :179–207, 2011.

[23] D. Koller and N. Friedman. *Probabilistic Graphical Models*. MIT Press, 2009.

[24] V. Kolmogorov, J. Thapper, and S. Zivny. The power of linear programming for general-valued CSPs. *SIAM J. Comput.*, 44(1) :1–36, 2015.

[25] C. Muise, S. McIlraith, J. Beck, and E. Hsu. Dsharp : Fast d-DNNF compilation with sharpSAT. In *Proc. of Canadian Conf. on AI*, pages 356–361, 2012.

[26] K. Pipatsrisawat and A. Darwiche. New compilation languages based on structured decomposability. In *Proc. of AAAI*, pages 517–522, 2008.

[27] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems : Hard and easy problems. In *Proc. of IJCAI*, pages 631–639, 1995.

[28] A. Schrijver. *Combinatorial Optimization : Polyhedra and Efficiency*, volume B. Springer, 2003.

[29] C. Sutton and A. McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4) :267–373, 2012.

[30] Z. Svikina and E. Tardos. Min-max multiway cut. In *Proc. of APPROX*, pages 207–218, 2004.

[31] Z. Svitkina and L. Fleischer. Submodular approximation : Sampling-based algorithms and lower bounds. *SIAM J. Comput.*, 40(6) :1715–1737, 2011.

[32] J. Thapper and S. Zivny. Necessary conditions for tractability of valued csps. *SIAM J. Discrete Math.*, 29(4) :2361–2384, 2015.

[33] J. Uckelman, Y. Chevaleyre, U. Endriss, and J. Lang. Representing utility functions via weighted goals. *Mathematical Logic Quaterly*, 55(4) :341–361, 2009.

[34] N. Wilson. Decision diagrams for the computation of semiring valuations. In *Proc. of IJCAI*, pages 331–336, 2005.

[35] S. Zivny. *The Complexity of Valued Constraint Satisfaction Problems*. Cognitive Technologies, 2012.

[36] S. Zivny, D. Cohen, and P. Jeavons. The expressive power of binary submodular functions. *Discrete Applied Mathematics*, 157(15) :3347–3358, 2009.

Amélioration des opérations sur MDD pour la création de modèle

Guillaume Perez **Jean-Charles Régin**

Université Nice-Sophia Antipolis, CNRS, I3S UMR 7271, 06900 Sophia Antipolis, France
guillaume.perez06@gmail.com jcregin@gmail.com

Résumé

Les MDD sont actuellement utilisés en PPC comme structure de données compressantes, bien souvent à partir d'un ensemble de tuples ou à partir d'un automate. Dans cet article nous montrons que les MDD peuvent aussi être utilisés pour modéliser directement des contraintes complexes afin de représenter rapidement et simplement des modèles avancés. Pour ce faire, nous introduisons de nouveaux constructeurs de MDD et proposons des algorithmes plus efficaces pour réaliser les operations courantes sur les MDD comme l'intersections et la difference symétrique ... Nous mettons en évidence l'avantage de notre approche sur un problème industriel de grande taille. Nous détaillons comment la modélisation directe de ce problème par des MDD puis l'utilisation d'opérateurs combinant ces MDD permet de le résoudre plus efficacement qu'avec un algorithme ad-hoc.

Abstract

MDDs, as used in CP, are compressed data structures, often defined from a set of tuples or from an automaton. In this paper we show that MDDs can also be used to model complex constraints, in order to build easily and rapidly advanced models. To do that we introduce new MDDs constructors and improve the existing ones, we propose efficient algorithms to perform classical operations like the union or the symmetric difference... We present the advantages of our method on a real industrial problem with huge domain size. We detail how to model this problem directly into MDDs and we experimentally show that our method outperforms the ad-hoc algorithm proposed for this problem.

1 Introduction

Les Multi-Valued Decision Diagrams (MDD) sont une structure de données de plus en plus utilisés en PPC. La raison de leur utilisation est qu'un MDD peut représenter de façon polynomial un nombre exponentiel de tuples.

En programmation par contraintes, les MDD sont utilisés pour la conversion de contraintes de table en MDD ou pour la modélisation d'autres contraintes (regular, slide...). Plusieurs propagateurs ont été définis [4, 8] et sont présents dans certains solveurs CP.

Les contraintes de table sont définies par la liste explicite des tuples valides. Elles permettent une définition simple de contraintes parfois complexes à définir autrement. Ces contraintes sont implémentées dans tous les solveur CP et sont fondamentales. Nous pouvons voir une table comme un MDD avec une racine d'ou partirai chaque tuple et qui se retrouverai au noeud final, autrement dit un MDD non compressé.

Dans cet article, nous donnons les outils aux constructeurs de modèle pour qu'ils puissent facilement définir des problèmes complexes comme ils l'auraient fait avec une table, mais avec des MDD, gagnant ainsi en espace.

Trois types d'opérations sont abordées, La création de MDD, la réduction de MDD et les opérations inter-MDD. Tout d'abord nous montrons comment construire un MDD depuis une contrainte basée sur un `automate` (régular...). Nous introduisons ensuite `Pattern`, un constructeur basé sur une fonction d'état qui stocke de l'information dans les noeuds durant la création. Ce constructeur permet de définir une plus large gamme de contraintes comme le knapsack, allDifferent ou les contraintes basées sur la programmation dynamique.

Pour la réduction, opération fondamentale pour un MDD qui fusionne les noeuds aillants les même arcs sortants (label, destination) et qui supprime les noeuds qui n'appartiennent pas à un chemin de la racine au noeud terminal vrai, plusieurs algorithmes ont été définis [3, 4]. Nous introduisons `pReduce`, le premier opérateur de réduction linéaire, avec une complexité en `O(n+m)` dans le pire des cas. Notre algorithme travaille niveau par niveau, de bas en haut, et utilise une procédure simple et originale pour fusionner et supprimer les noeuds.

Finalement, nous introduisons un opérateur générique permettant le calcul d'opérations inter-MDD tel que l'intersection, l'union, la difference symétrique... L'algorithme est basé sur des opérations entre les graphes, il parcours le graphe en BFS sur les deux MDD en même temps et etudie les noeuds par couple, un noeud de chaque MDD à la fois.

Cette article est une version résumé de l'article [9]. Nous donnons ici les idées des algorithmes, et invitons les lecteurs à lire la version originale pour plus de precision. Nous décrivons ici l'algorithme de reduction pReduce, une des deux contributions majeures du papier original.

2 Définitions

Un multi-valued decision diagram (MDD) est une structure de données pour représenter une fonction discrète. C'est une extension multi-valeur des BDDs [3].

Un MDD, comme utilisé en CP, [1, 5, 6, 2], est un Directed Acyclic Graph (DAG) avec une racine unique, utilisé pour représenter une fonction $f : \{0...d-1\}^r \rightarrow \{true, false\}$, pour un entier donné d. Pour r variables, la représentation par un DAG contient r niveaux, tel que chaque variable est représentée par un unique niveau. Chaque noeud d'un niveau possède au plus d arcs sortants vers les noeuds du niveau suivant du graphe. Chaque arc possède une étiquette correspondant à sa valeur. Le niveau final est représenté par le noeud terminal true. Il y a une équivalence entre $f(v_1, ..., v_r) = true$ et l'existence d'un chemin du noeud racine au noeud terminal true et dont les arcs sont étiquetés par $v_1, ..., v_r$. Les noeuds sans aucun arc sortant ou arc entrant sont supprimés.

Dans une contrainte MDD, le MDD modélise l'ensemble des tuples qui satisfont la contrainte, comme dans une contrainte définie en extension. Chaque variable du MDD correspond à une variable de la contrainte. Un arc associé à une variable du MDD correspond à une valeur de la variable correspondante de la contrainte.

Par commodité, nous allons noter d le nombre maximum de valeurs dans le domaine d'une variable et un arc sortant de x vers y étiqueté par v sera noté (x, v, y). Nous noterons aussi par $\omega^+(x)$ l'ensemble des arcs sortants de x.

Un exemple de MDD est donné en Figure 1. Ce MDD représente les tuples {a,a}, {a,b}, {c,a}, {c,b} et {c,c}. Pour chaque tuple, il y a un chemin du noeud racine (0) vers le noeud terminal (tt) dont les arcs sont étiquetés par les valeurs du tuple.

Dans cet article, nous considérerons que les ω^+ sont des listes triées, cela implique que tous les algorithmes de création, réduction et d'opération doivent maintenir cette propriété. Cela permet par exemple d'effectuer rapidement certaines operations comme l'intersection de deux noeuds.

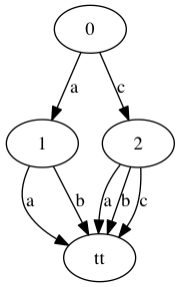


FIGURE 1 – Un MDD representant l'ensemble de tuples {{a,a},{a,b},{c,a},{c,b},{c,c}}

3 pReduce

La réduction d'un MDD est une des opérations les plus importantes. Elle consiste à fusionner les noeuds équivalents, i.e. les noeuds possédant le même ensemble ω^+ en ne considérant que les deux dernières composantes du triplet (x, v, y). Généralement, un algorithme de réduction fusionne les noeuds jusqu'à arriver à un point fixe. Notons que l'opération de réduction ne peut pas augmenter le nombre d'arcs ni de noeuds.

Cheng and Yap [4] ont proposé un algorithme pour reduire un MDD, Pour cela ils parcourent le MDD en effectuant une DFS avec un traitement en post-visite. Ce traitement est la recherche du noeud équivalent au noeud courrant dans un dictionnaire. Pour effectuer ce traitement, les auteurs utilisent une hash-map, ou un radix tree. dans les deux cas, la complexité n'est pas linéaire. Soit θ la complexité de la structure de donnée choisie, la complexité est de $O(n * d * \theta)$, avec n le nombre de noeud et d la taille du domaine.

Nous proposons une méthode de réduction d'un MDD linéaire en temps et en espace. Plutôt que de chercher, pour chaque noeud, s'il doit être fusionné ou non, nous allons chercher la première différence entre les noeuds, c'est à dire le premier arc différent entre les noeuds. Pour cela, nous allons traiter les noeuds par paquets, et diviser ces paquets en fonction de leurs arcs sortants. Précisement, nous effectuons une BFS en bottom-up, prenons tous les noeuds d'un niveau et les intégrons à un paquet, puis nous divisons iterativement ce paquet en sous-paquet en fonction du prochain arc sortant des noeuds, les noeuds ayant le même arc (étiquette, destination) resterons dans le même paquet. Cela peut être fait en deux étapes, la première divisant les paquets en fonction de la valeur de l'étiquete de l'arc, la suivante divisant en fonction du noeud destination. Au moment de la i^{eme} division d'un paquet, les i premiers arcs des listes ω^+ des noeuds du paquet sont les mêmes. Lorsqu'un paquet contient un unique noeud, il n'est plus utile de continuer à traiter ses arcs sortants. Lorsqu'il n'y a plus d'arc à traiter, cela signifie que la totalité de leurs arcs sont

les mêmes, les noeuds du paquet sont donc équivalents et peuvent être fusionnés. La complexité finale de cet algorithme est de $O(n+m+d)$ en temps et en espace, avec m le nombre total d'arc du MDD, sachant que $m \leq n * d$.

4 Conclusion

Dans ce resumé, nous avons décrit le fonctionnement de l'algorithme de réduction d'un MDD, et montré que sa complexité est linéaire. Dans la version originale, les algorithmes introduits permettent de travailler avec des MDD très gros, de l'ordre de plusieurs millions de noeuds et de centaines de millions d'arcs. De plus les algorithmes définis n'ont plus la taille du domaine comme facteur en complexité par noeud (spatiale ou temporelle), cela nous permet donc de travailler sur des instances avec des domaines très grand (plus de 11 000 valeurs).

Références

[1] Henrik Reif Andersen, Tarik Hadzic, John N. Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *CP*, pages 118–132, 2007.

[2] David Bergman, Willem Jan van Hoeve, and John N. Hooker. Manipulating mdd relaxations for combinatorial optimization. In *CPAIOR*, pages 20–35, 2011.

[3] Randal E Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8) :677–691, 1986.

[4] K. Cheng and R. Yap. An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15, 2010.

[5] Tarik Hadzic, John N. Hooker, Barry O'Sullivan, and Peter Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In *CP*, pages 448–462, 2008.

[6] Samid Hoda, Willem Jan van Hoeve, and John N. Hooker. A systematic approach to mdd-based constraint programming. In *CP*, pages 266–280, 2010.

[7] A. Papadopoulos, P. Roy, and F. Pachet. Avoiding plagiarism in markov sequence generation. In *Proceeding of the Twenty-Eight AAAI Conference on Artificial Intelligence*, pages 2731–2737, 2014.

[8] Guillaume Perez and Jean-Charles Régin. Improving GAC-4 for table and MDD constraints. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 606–621, 2014.

[9] Guillaume Perez and Jean-Charles Régin. Efficient operations on mdds for building constraint programming models. *International Joint Conference on Artificial Intelligence, IJCAI-15, Argentina*, 2015.

[10] Guillaume Perez and Jean-Charles Régin. Relations between mdds and tuples and dynamic modifications of mdds based constraints. *arXiv preprint arXiv :1505.02552*, 2015.

[11] L. Perron. Or-tools. In *Workshop "CP Solvers : Modeling, Applications, Integration, and Standardization"*, 2013.

[12] G. Pesant. A regular language membership constraint for finite sequences of variables. In *Proc. CP'04*, pages 482–495, 2004.

Structures de données persistantes pour le backtracking et le parallélisme en PPC

Julien Vion
LAMIH CNRS UMR 8201, UVHC, 59313 Valenciennes
julien.vion@univ-valenciennes.fr

Résumé

Nous présentons les structures de données utilisées par le solveur *Concrete* pour représenter l'état d'un problème au cours de la recherche. L'utilisation de structures purement fonctionnelles permet de réaliser très facilement un backtracking non chronologique ou une résolution en parallèle.

Abstract

We present the data structures used by the *Concrete* constraint solver to represent the state of a problem during search. The use of purely functional data structures enable us to implement non-chronological backtracking and parallel resolution easily.

Les *structures de données persistantes* [4] permettent de conserver les anciennes versions d'elles-mêmes lorsqu'elles sont modifiées. Elles sont fortement apparentées aux structures de données *immutables*, c'est-à-dire que leur contenu ne peut pas être modifié. À la place, lors d'une opération de modification (l'ajout d'un élément, par exemple), une nouvelle structure de données est créée, contenant les données mises à jour. La structure de données persistante la plus simple est la liste chainée, représentée par une paire $(tête, queue)$, la tête étant le premier élément de la liste et la queue la liste des autres éléments, ou la liste vide $\langle\rangle$. Prenons l'exemple de la liste chainée $xs = \langle 1, 2, 3\rangle = (1, (2, (3, \langle\rangle)))$. Ajouter la valeur 0 en tête de la liste ne nécessite pas de dupliquer d'information, mais simplement de créer une nouvelle paire $xs' = (0, xs)$. La liste xs elle-même n'est pas modifiée et peut toujours être accédée. La plupart des structures de données arborescentes peuvent être facilement adaptées pour être persistantes.

Ces structures de données ont été grandement développées dans les années 1990 conjointement au développement des langages fonctionnels [10]. Les langages fonctionnels modernes comme Haskell, OCaml, Scala ou Clojure incluent tous une librairie de structures de données persistantes dont les performances sont comparables aux structures traditionnelles.

Les structures persistantes permettent d'implanter très facilement un backtracking. Par exemple, pour effectuer une recherche en profondeur d'abord (e.g., MAC), on maintient simplement une pile contenant les différentes versions des structures de données à chaque niveau de l'arbre de recherche. Un autre avantage des structures immutables est qu'elles permettent d'implémenter très facilement un backtracking non chronologique, ou encore une parallélisation des algorithmes de recherche : il n'est plus nécessaire de synchroniser l'accès aux informations si celles-ci ne peuvent pas être modifiées. Les stratégies de type *Embarassingly Parallel Search* [11] ou autres cousines du *MapReduce* nécessitent de dupliquer l'état complet du problème pour chaque sous-problème à évaluer. Une structure persistante permet de ne mémoriser que les éléments modifiés de chaque sous-problème.

Le solveur *Concrete* [13] est entièrement développé en Scala. Dans cet article, nous présentons les structures de données choisies pour représenter les domaines des variables et les états des contraintes. Il s'agit d'une première ébauche de classification de diverses structures de données, et nous espérons que ce retour d'expérience sera utile aux développeurs de solveurs désireux d'exploiter au mieux les fonctionnalités apportées par les langages fonctionnels.

1 Représentation du problème

Un problème de satisfaction de contraintes est généralement représenté par un triplet $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, avec $\mathcal{X} = \{X_1, \ldots, X_n\}$ les variables, $\mathcal{D} = \{D_1, \ldots, D_n\}$ les domaines de définition de ces variables et $\mathcal{C} =$

$\{C_1, \ldots, C_e\}$ les contraintes. D_i est le domaine de la variable X_i. $(\mathcal{X}, \mathcal{C})$ forme un hypergraphe dont les variables sont les sommets et les contraintes les arêtes. On note vars(C) les variables impliquées par une contrainte C. Chaque contrainte définit l'ensemble des instanciations autorisées des variables qu'elle implique. On associe à chaque contrainte un *propagateur* qui détecte des valeurs qui n'apparaissent dans aucune instanciation autorisée [5]. Certains propagateurs peuvent détecter que toutes les instanciations des variables sont autorisées : la contrainte est alors dite « universelle » ou *entailed* et peut être désactivée.

Le modèle de résolution général des CSP est une recherche en profondeur d'abord. À chaque nœud de l'arbre, on effectue une hypothèse qui réduit le domaine d'une voire plusieurs variables, puis on effectue un filtrage en exécutant les propagateurs des contraintes jusqu'au point fixe. Pour des raisons de performances, les propagateurs sont généralement incrémentaux : en effet, tout au long d'une branche de l'arbre de recherche, chaque propagateur est appelé de très nombreuses fois. À chaque appel, les domaines des variables ont uniquement « diminué », c'est-à-dire qu'ils ont perdu des valeurs. Il est généralement possible d'utiliser cette propriété pour améliorer grandement les performances, voire la complexité des algorithmes de propagation [9, 2]. En général, cela nécessite d'associer un *état* à chaque contrainte, qui devra être maintenu au cours de la recherche.

Pour réaliser la recherche en profondeur d'abord, il faut être capable de *backtracker*, c'est-à-dire annuler une hypothèse et ses conséquences quand elle conduit à une contradiction. Pour ce faire, le solveur *Concrete* maintient une pile d'états au cours de la recherche. Chaque état de problème contient trois ensembles d'informations : • le contenu du domaine de chaque variable, • l'état de chaque contrainte si nécessaire, et • pour chaque variable, l'ensemble des contraintes actives qui l'impliquent.

2 Représentation des domaines

Concrete supporte les domaines basés sur les booléens et les entiers relatifs bornés. La plus petite valeur d'un domaine D est notée $\underline{D}$ et la plus grande $\overline{D}$. Les domaines booléens (également utilisés pour représenter les domaines entiers inclus dans $\{0, 1\}$) ne présentent pas de difficulté particulière. Un domaine booléen ne peut prendre que quatre états : *vide*, *vrai*, *faux* ou *indéterminé*. Chacun est représenté par un objet singleton immutable.

Sur les autres domaines entiers, les deux opérations les plus importantes sont 1. l'énumération des éléments à partir d'une valeur arbitraire, et 2. le filtrage selon un prédicat. Ces deux opérations suffisent à implanter un algorithme de filtrage général comme AC-3 ou une de ses variantes. Le contrôle très rapide de la présence d'une valeur ou la possibilité de retirer une valeur arbitraire est également souvent utile pour implanter des propagateurs spécifiques. Toute structure traditionnellement utilisée pour réaliser un ensemble mathématique, comme une table de hash, pourrait convenir. Les solveurs de contraintes utilisent souvent une structure de données backtrackable par *« trailing »* [7] ou un *Sparse Set* [3] pour implémenter de tels domaines. Ces structures ne sont que partiellement persistantes : il n'est pas possible de modifier une ancienne version de la structure de manière transparente. Elles ne peuvent pas être utilisées dans le cadre du parallélisme.

Pour réaliser un filtrage aux bornes, la possibilité d'effectuer une « coupe », i.e., supprimer rapidement tous les éléments supérieurs ou inférieurs à une valeur donnée est également importante. Cette observation conduit à observer les domaines comme des ensembles ordonnés. Les ensembles ordonnés sont traditionnellement implémentés sous forme d'arbres équilibrés, mais dans ce cas le test de présence et la suppression d'un élément sont en $O(\log |D|)$, et le filtrage en $O(|D| \log |D|)$.

Les opérations d'union et d'intersection entre deux domaines ou ensembles permettent d'exprimer facilement de nombreuses propriétés de consistance. Par exemple, on peut considérer la recherche du support d'une valeur $a \in \mathrm{dom}(X)$ dans la contrainte C portant sur (X, Y) comme une intersection entre les supports de X pour C et le domaine de Y. Cette observation nous a conduit par le passé à développer des algorithmes basés sur une représentation par vecteurs de bits des domaines [8], qui ont été depuis repris par de nombreux solveurs. Les contraintes *min/max* ou *element* peuvent également être partiellement exprimées par des opérations d'union/intersection.

Les domaines sur les nombres entiers dans *Concrete* sont polymorphiques. Sous une même interface, ils peuvent être réalisés soit par un objet représentant l'ensemble vide, soit par une valeur singleton pour représenter les variables affectées, soit par un intervalle de nombres entiers, soit par des vecteurs de bits. Ces derniers sont la représentation la plus générale.

La représentation par intervalles est utilisée si toutes les valeurs entières comprises entre $\underline{D}$ et $\overline{D}$ sont présentes dans D (on dit que D est *convexe*). Dans ce cas, seule la paire $(\underline{D}, \overline{D})$ est stockée en mémoire. L'espace de stockage est minimal, et les opérations de coupe ainsi que la suppression du plus petit ou du plus grand élément peuvent être réalisées en temps constant. En cas de suppression d'un élément au milieu de l'intervalle, un vecteur de bits est créé et retourné. Au contraire, si

après une modification de vecteur de bits on constate que $\overline{D} - \underline{D} = |D| - 1$, alors D est convexe et est représenté par un intervalle.

Un vecteur de bits est un tableau d'entiers naturels en base 2, représentant chacun b bits (généralement 64). On peut tester en $O(1)$ la présence d'un bit par des opérations de rotation et masquage bit-à-bit disponibles à bas niveau dans la plupart des langages de programmation[1]. Les bits sont indexés à partir de 0. Pour représenter un ensemble de valeurs arbitraires de manière compacte, on associe au vecteur de bits une fonction bijective associant à chaque index la valeur correspondante, et vice-versa. Cette fonction peut être réalisée par une structure de données ad-hoc, mais plus simplement, nous associons à chaque domaine un « offset » tel que pour un domaine D, le bit d'index $i - \mathit{offset}$ est à *vrai* ssi $i \in D$. Cela permet de représenter des valeurs négatives, ou optimiser la représentation d'un domaine commençant par une « grande » valeur. Cette représentation est en $O(\overline{D} - \underline{D})$, ce qui n'est pas optimal mais le surcout est rarement mesurable en pratique. Pour réaliser les opérations d'union et intersection entre deux domaines, il faut également que les associations index/valeur des opérandes correspondent. Les opérations de rotation à gauche ou à droite permettent d'établir la correspondance des offsets de manière efficace. De plus, *Concrete* conserve le dernier ajustement d'offset demandé en cache, notamment pour réaliser efficacement la technique décrite dans [8]. Ensuite, on peut réaliser l'opération d'union ou d'intersection par paquets de b bits à l'aide des opérateurs bit-à-bit du langage hôte.

Il est possible que $\underline{D} > \mathit{offset}$ suite à la suppression des premiers éléments du domaine, mais si $\underline{D} - \mathit{offset} \geq b$, un décalage est automatiquement réalisé et l'offset est mis à jour. Si $\overline{D} - \mathit{offset} < b$, un seul entier est nécessaire pour représenter le domaine. Nous utilisons le polymorphisme pour optimiser ce cas particulier.

Le principal inconvénient de la représentation par vecteurs de bits est qu'elle n'est pas naturellement persistante ou immutable. Cependant, sa relative compacité en fait un bon candidat pour des stratégies de recopie [12]. Pour obtenir un comportement véritablement persistant, il faut réaliser une copie du domaine à chaque fois que celui-ci est modifié. Retirer un seul élement du domaine nécessite de recopier l'ensemble du tableau (soit $\lceil \frac{\overline{D} - \underline{D}}{b} \rceil$ entiers de b bits à recopier). Si on retire ainsi chaque élément un par un, on peut totaliser $O(|D|^2)$ opérations. Lorsque l'on fait appel à la fonction de filtrage par un prédicat de cout $O(f)$ (cas le plus courant), il est possible de ne réaliser la copie qu'une seule fois et ainsi de faire l'ensemble du traitement de manière optimale en $O(|D|\, f)$. Cette fonctionnalité est importante car elle garantit de ne pas dégrader la complexité des algorithmes de propagation. L'opération de coupe inférieure jusqu'à la valeur $c \leq \overline{D}$ nécessite de recopier $\lceil \frac{c - \underline{D}}{b} \rceil$ entiers de b bits, et vice versa pour la coupe supérieure. Cette complexité est identique (au facteur b près) aux structures traditionnelles avec *trailing*.

[1]Certains langages comme Java ou Scala intègrent directement les vecteurs de bits dans leur bibliothèque standard.

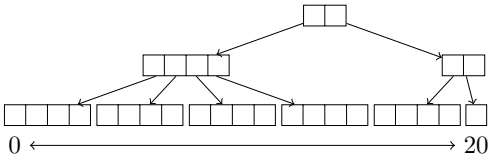


FIG. 1 : Un vecteur immutable de 21 éléments ($m = 4$)

3 Représentation de l'état du problème

La question de rendre persistante l'état global du problème reste entière. Bien sûr, on peut envisager de recopier l'état du domaine de chaque variable et de chaque contrainte à chaque nœud de l'arbre de recherche. Si les structures de données des domaines et des contraintes sont déjà persistantes, cela nécessite un traitement en $O(n + e)$ à chaque nœud. Le plus souvent, l'état d'une minorité de variables et de contraintes sont modifiés après un filtrage. Il est alors indispensable, pour des problèmes de grande taille, de ne sauvegarder que ce qui est nécessaire. On peut alors utiliser une technique de *trailing* classique pour gérer le backtrack. Cette technique n'est pas applicable si plusieurs traitements se font en parallèle.

Une alternative séduisante consiste à utiliser une structure de données persistante pour stocker l'état global du problème. Une référence sur une version de la structure de données est alors suffisante pour sauvegarder et restaurer l'état du problème à tout moment en temps constant, y compris dans le cadre de traitements en parallèle. Pour le stockage de l'état des domaines, nous avons besoin d'une structure de données capable d'associer un domaine à chaque variable, et permettant une lecture et mise à jour rapide dans un cadre persistant. Si les variables sont indexées de 0 à $n - 1$, on peut utiliser un simple tableau de domaines pour réaliser cette association. Cependant, comme évoqué précédemment, il faut utiliser une structure moins naïve qu'un tableau pour ne pas avoir à copier l'ensemble des domaines à chaque nœud, mais uniquement ceux qui ont été modifiés, tout en gardant de bonnes performances en lecture. La même remarque s'applique pour les contraintes et leurs états.

La structure de *vecteur immutable* introduite par le

langage de programmation Clojure [6] et inspirée par les *Ideal Hash Tries* [1] réalise un bon équilibre entre les performances de lecture et de modification. Les vecteurs immutables sont implantés sous forme d'arbres m-aires. La figure 1 illustre un vecteur immutable quaternaire. Chaque nœud est un tableau d'au plus m éléments. Les feuilles contiennent les données, les nœuds intermédiaires forment un chemin depuis la racine jusqu'à l'élément recherché. Pour un vecteur de taille n, la profondeur de l'arbre est de $O(\log_m n)$. Pour modifier un élément, on recopie chacun des nœuds conduisant à cet élément, soit $O(m \log_m n)$ opérations. Le choix typique de $m = 32$ réalise un bon compromis entre la vitesse de consultation et la vitesse de modification sur les machines modernes, en prenant notamment en compte les caches matériels et les optimisations sur les calculs de modulo. Dans le langage Java, il n'est pas possible de créer un tableau de plus de 2^{31} éléments. Cette restriction s'applique sur la plupart des structures de données du langage et est rarement remise en cause. Le vecteur immutable représentant un tel tableau n'aurait qu'une profondeur de 7 avec $m = 32$. En pratique, les opérations sur les vecteurs immutables peuvent être considérées de complexité constante.

4 Performances et conclusion

Une fois les structures de données définies, l'implémentation de n'importe quel algorithme de propagation et de recherche est relativement simple, à condition que tous les états des contraintes soient persistants. Certaines structures partiellement persistantes comme les *Sparse Sets*, voire non persistantes comme les *résidus* peuvent également être utilisées sous conditions (synchronisation, backtracking chronologique…) Cependant, bien que de complexité constante ou presque, les surcouts que représentent les recopies de domaine et parcours des vecteurs immutables ne sont pas nécessairement négligeables.

La longueur de cet article ne permet pas de rendre compte de benchmarks détaillés pour chaque structure de données. Nous avons comparé les performances des solveurs *CSP4J* [14] et *Concrete 3*. Ces deux solveurs utilisent les mêmes algorithmes de branchement et de propagation, mais *CSP4J* était entièrement basé sur des structures de *trailing* alors que *Concrete 3* se base sur des structures persistantes. L'impact du cout des vecteurs immutables est pratiquement négligeable devant des algorithmes de propagation de complexité au moins quadratique. Cependant, *Concrete 3* peut se révéler jusqu'à deux fois plus lent que *CSP4J* sur des propagateurs très rapides (contraintes binaires ou arithmétiques aux bornes). Des benchmarks plus complets sont en cours de réalisation. L'intérêt de ces structures de données est clairement dans leur simplicité d'utilisation, et les perspectives pour réaliser facilement et sans surcout des algorithmes de backtrack non-chronologiques ou parcourus en parallèle.

Références

[1] P. Bagwell. *Ideal Hash Trees*. Rapp. tech. EPFL, 2001.

[2] C. Bessière, J.-C. Régin, R.H.C. Yap et Y. Zhang. "An Optimal Coarse-Grained Arc Consistency Algorithm". In : *Artificial Intelligence* 165.2 (2005), p. 165–185.

[3] P. Briggs et L. Torczon. "An Efficient Representation for Sparse Sets". In : *ACM Letters on Programming Languages and Systems* 2.1–4 (1993), p. 59–69.

[4] J.R. Driscoll, N. Sarnak, D. Sleator et R. Tarjan. "Making Data Structures Persistent". In : *J. of Computer and System Science* 38 (1989), p. 86–124.

[5] P. van Hentenryck, Y. Deville et CM. Teng. "A Generic AC Algorithm and its Specializations". In : *Artificial Intelligence* 57 (1992), p. 291–321.

[6] R. Hickey. *The Clojure Programming Language*. `http://clojure.org/`. 2006.

[7] C. Lecoutre et R. Szymanek. "GAC for Positive Table Constraints". In : *Proc. CP'06*. Nantes, France, 2006, p. 284–298.

[8] C. Lecoutre et J. Vion. "Enforcing AC using Bitwise Operations". In : *Constraint Programming Letters* 2 (2008), p. 21–35.

[9] R. Mohr et T.C. Henderson. "Arc and Path Consistency Revisited". In : *Artificial Intelligence* 28 (1986), p. 225–233.

[10] C. Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1998.

[11] J.-C. Régin, M. Rezgui et A. Malapert. "Embarrasingly Parallel Search". In : *Proc. 19th Itl Conference on Principles and Practice of CP*. 2013, p. 596–610.

[12] C. Schulte. "Comparing Trailing and Copying for Constraint Programming". In : *Proceedings of ICLP'99*. 1999, p. 275–289.

[13] J. Vion. *Concrete : a CSP Solving API for the JVM*. `http://github.com/concrete-cp`. 2006–2014.

[14] J. Vion. *Constraint Satisfaction Problem For Java*. `http://cspfj.sourceforge.net/`. 2006–2012.

Programmation linéaire mixte et programmation par contraintes pour un problème d'ordonnancement à contraintes énergétiques

Margaux Nattaf[1] **Christian Artigues**[1] **Pierre Lopez**[1]

[1] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
{nattaf,lopez,artigues}@laas.fr

Résumé

Nous considérerons un problème d'ordonnancement cumulatif dans lequel les tâches ont une durée et un profil de consommation de ressource variable. Ce profil, qui peut varier en fonction du temps, est une variable de décision du problème dont dépend la durée de la tâche associée. Pour ce problème NP-difficile, nous présentons un modèle de programmation par contraintes et un modèle de programmation linéaire en nombres entiers (PLNE). De plus, des inégalités valides déduites de la programmation par contraintes viennent renforcer le PLNE. Ces modèles sont ensuite comparés par le biais d'expérimentations.

Abstract

We study a cumulative scheduling problem where a task duration and resource consumption are not fixed. The consumption profile of the task, which can vary continuously over time, is a decision variable of the problem and the task duration depends on this profile. For the discrete case, the paper presents a mixed integer linear program as well as a constraints programming model. Furthermore, valid inequalities deduced from constraint programming are also provided. Both models are then compared through computational experiments.

1 Introduction

Nous étudions un problème d'ordonnancement avec ressource continue et contraintes énergétiques, le Continuous Energy-Constrained Scheduling Problem (CECSP). Dans ce problème, un ensemble de tâches $\mathcal{A} = \{1, \ldots, n\}$ utilisant une ressource continue et cumulative de capacité limitée B doit être ordonnancé. La quantité de ressource nécessaire à l'exécution d'une tâche n'est pas fixée mais - le profil de consommation de cette dernière est une fonction $b_i(t)$ définie pour tout $t \in \mathbb{R}$ [1] - doit être déterminé. Une fois la tâche commencée et jusqu'à sa date de fin, la fonction $b_i(t)$ doit être comprise entre une valeur maximale, b_i^{max}, et minimale, b_i^{min}.

De plus, la consommation, à un instant t, d'une partie de la ressource permet la production d'une certaine quantité d'énergie et, une tâche finit lorqu'elle a reçu une énergie W_i. Cette énergie est calculée par le biais d'une fonction de rendement f_i, propre à chaque tâche. Dans cet article, ces fonctions sont supposées continues, croissantes, affines et peuvent être exprimées de la manière suivante :

$$f_i(b) = \begin{cases} 0 & \text{si } b = 0 \\ a_i * b + c_i & \text{si } b_i^{min} = 0 \text{ et } b \in]b_i^{min}, b_i^{max}] \\ a_i * b + c_i & \text{si } b_i^{min} \neq 0 \text{ et } b \in [b_i^{min}, b_i^{max}] \end{cases}$$

avec $a_i > 0$ et $c_i \geq -a_i * b_i^{min}$ pour s'assurer que $f_i(b) \geq 0,\ \forall b \in [b_i^{min}, b_i^{max}]$.

Dans la suite, nous dénotons par $\underline{s_i}$ et $\overline{s_i}$ la date de début au plus tôt et au plus tard de i et par $\underline{e_i}$ et $\overline{e_i}$ la date de fin au plus tôt et au plus tard de i.

Pour trouver une solution pour le CECSP, nous devons déterminer, pour chaque tâche $i \in \mathcal{A}$, sa date de début s_i, sa date de fin e_i et sa fonction d'allocation de ressource $b_i(t)$, $\forall t \in \mathcal{T} = [\min_{i \in \mathcal{A}} \underline{s_i}, \max_{i \in \mathcal{A}} \overline{e_i}]$. De plus, ces variables doivent satisfaire les contraintes suivantes :

$$\underline{s_i} \leq s_i < e_i \leq \overline{e_i} \qquad \forall i \in \mathcal{A} \tag{1}$$

$$b_i^{min} \leq b_i(t) \leq b_i^{max} \qquad \forall i \in \mathcal{A},\ \forall t \in [s_i, e_i] \tag{2}$$

$$b_i(t) = 0 \qquad \forall i \in \mathcal{A},\ \forall t \notin [s_i, e_i] \tag{3}$$

$$\int_{s_i}^{e_i} f_i(b_i(t))dt = W_i \qquad \forall i \in \mathcal{A} \tag{4}$$

$$\sum_{i \in \mathcal{A}} b_i(t) \leq B \qquad \forall t \in \mathcal{T} \tag{5}$$

1. Le domaine de définition de la fonction peut être réduit mais, pour faciliter les notations, nous supposons qu'elle est définie pour tout $t \in \mathbb{R}$.

L'objectif auquel nous nous sommes intéressés est la minimisation de la consommation totale de la ressource. Dans [4], les auteurs montrent que trouver une solution admissible pour le CECSP est déjà un problème NP-complet.

De plus, une instance ayant des données seulement entières peut n'avoir que des solutions à valeurs dans $\mathbb{R}$ [4]. Cependant, une dilatation de l'instance, i.e. multiplier les données par un certain coefficient α, permet de palier à ce problème. De ce fait et dans le but de résoudre des instances entières, nous nous sommes intéressés, dans un premier temps, à la version discrète du CECSP, le DECSP (Discrete Energy Constrained Scheduling Problem). Dans ce problème, toutes les données sont supposées entières et les domaines de chaque variable ne contiennent que des valeurs entières, i.e. $s_i,\ e_i,\ b_i(t) \in \mathbb{N}$ et $b_i(t)$ est défini $\forall t \in \mathcal{T}_\mathcal{D} = \{\min_{i\in\mathcal{A}} \underline{s_i}, \ldots, \max_{i\in\mathcal{A}} \overline{e}_i\}$.

Pour ce problème, nous présentons un modèle de programmation par contraintes (PPC) permettant l'utilisation des algorithmes de propagation mis en place pour la contrainte cumulative, notamment [3]. Un modèle de programmation linéaire en nombres entiers (PLNE) est aussi présenté. Ce modèle est ensuite renforcé à l'aide d'inégalités valides déduites du raisonnement énergétique [2]. Ces deux modèles sont ensuite testés sur des instances à données entières, avec et sans dilatation.

2 Modèle de programmation par contrainte

Pour modéliser le DECSP à l'aide de la PPC, nous divisons chaque tâche i en deux sous-tâches i_{min} et i_{preem}. La première, i_{min}, est une tâche ayant une consommation de ressource fixe, égale à b_i^{min}, et une durée variable p_i. Cette tâche représente la quantité de ressource obligatoirement consommée par une activité durant son exécution, i.e. b_i^{min}. La seconde, i_{preem} est une tâche préemptive optionnelle, consommant une quantité variable de ressource comprise entre 0 et $b_i^{max} - b_i^{min}$ et devant s'exécuter en même temps que i_{min}. Cette tâche est elle-même divisée en sous-tâches $i^\ell_{preem},\ \ell \in \{1, \ldots, e_i - s_i\} = \mathcal{L}_i$. Notons que $|\mathcal{L}_i| = e_i - s_i \leq \lceil \frac{W_i}{f_i(b_i^{min})} \rceil$.

Exemple 1. *Considérons la tâche possédant les attributs suivant :* $\underline{s_i} = 0$, $\overline{e_i} = 6$, $W_i = 28$, $b^{min} = 1$, $b^{max} = 5$ *et* $f_i(b) = 2b+1$. *La Figure 1 présente un ordonnancement de cette tâche (à droite) et l'ordonnancement correspondant donné par le modèle (à gauche) avec* $b_{i^2_{preem}} = 0$.

Le problème du DECSP peut alors être formulé à l'aide des variables :

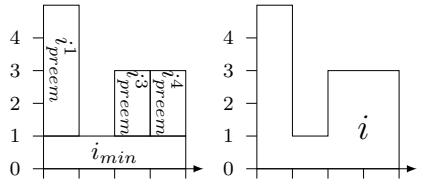


FIGURE 1 – Exemple de solution du modèle PPC.

— $i_{min} = \{s_{i_{min}}, e_{i_{min}}, b_{i_{min}}, p_{i_{min}}\},\ \forall i \in \mathcal{A}$
— $i^\ell_{preem} = \{s_{i^\ell_{preem}}, e_{i^\ell_{preem}}, b_{i^\ell_{preem}}, p_{i^\ell_{preem}}\}, \forall i \in \mathcal{A},\ \ell \in \mathcal{L}_i$.

et des contraintes :

1. $\forall (i,l) \in \mathcal{A} \times \mathcal{L}_i :\ e_{i^\ell_{preem}} = s_{i^{\ell+1}_{preem}}$
2. $\forall i \in \mathcal{A} :\ s_{i_{min}} = s_{i^1_{preem}}$ et $e_{i^{|\mathcal{L}_i|}_{preem}} = e_{i_{min}}$
3. $\forall t \in \mathcal{T} :$ $\sum_{i\in\mathcal{A}} \sum_{t\in[s_{i_{min}}, e_{i_{min}}[} b_{i_{min}} + \sum_{(i,l)\in\mathcal{A}\times\mathcal{L}_i} \sum_{t\in[s_{i^\ell_{preem}}, e_{i^\ell_{preem}}[} b_{i^\ell_{preem}} \leq B$
4. $\forall i \in \mathcal{A} :\ \sum_{l\in\mathcal{L}_i} \left(f_i(b_{i^\ell_{preem}})(e_{i^\ell_{preem}} - s_{i^\ell_{preem}}) \right) + f_i(b_{i_{min}})(e_{i_{min}} - s_{i_{min}}) \geq W_i$

La première contrainte permet d'ordonner les sous-tâches de i_{preem}. Ceci dans le but de faciliter la modélisation des autres contraintes. La seconde contrainte modélise le fait que i_{preem} commence et finit en même temps que i_{min}. La troisième contrainte assure que la capacité de la ressource n'est pas excédée en sommant, à un instant t, les consommations minimales des tâches en cours ainsi que les consommations des sous-tâches préemptives en cours. Enfin, la quatrième contrainte permet de s'assurer que chaque tâche reçoit au moins l'énergie requise W_i.

Un des avantages de cette formulation est qu'elle permet l'utilisation des algorithmes de propagation mis en places pour la contrainte cumulative tels que le time-table classique [1], disjonctif [3] ou associé au edge-finding [5], le raisonnement disjonctif [1], ou encore le raisonnement énergétique [2]. Cependant, certains de ces raisonnements peuvent être adaptés pour prendre en compte l'ensemble du problème. C'est le cas, par exemple, du raisonnement énergétique détaillé ci-dessous.

2.1 Raisonnement énergétique

Ce paragraphe présente un algorithme de propagation pour le DECSP basé sur le raisonnement énergétique défini pour le CECSP [4]. L'adaptation de ce raisonnement au cas discret est quasi-directe. Cependant, nous rappelons les bases de celui-ci car nous l'utiliserons dans la suite pour déduire des inégalités valides pour le PLNE.

Le principe du raisonnement énergétique est de comparer la quantité de ressource disponible dans un inter-

valle avec la quantité minimale de ressource consommée par toutes les tâches dans cet intervalle.

Les configurations pour lesquelles la quantité de ressource requise par une tâche i dans l'intervalle $[t_1, t_2[$ est minimale correspondent toujours à une configuration où la tâche reçoit le maximum d'énergie possible, i.e. est ordonnancée à b_i^{max}, en dehors de $[t_1, t_2[$, tout en respectant les contraintes (1)–(5). Ceci correspond donc à une des configurations suivantes :

— la tâche est calée à gauche : ordonnancée à b_i^{max} durant $[\underline{s_i}, t_1[$;
— la tâche est calée à droite : ordonnancée à b_i^{max} durant $[t_2, \overline{e_i}[$;
— la tâche est centrée : ordonnancée à b_i^{max} durant $[\underline{s_i}, t_1[\cup[t_2, \overline{e_i}[$ ou ordonnancée à b_i^{min} durant $[t_1, t_2[$.

En effet, dans le dernier cas, il peut arriver qu'ordonnancer la tâche à b_i^{max} dans $[\underline{s_i}, t_1[\cup[t_2, \overline{e_i}[$ implique que la quantité d'énergie restant à apporter à la tâche dans $[t_1, t_2[$ ne soit pas suffisante pour ordonnancer la tâche à b_i^{min} durant $[t_1, t_2[$. Or, ceci impliquerait une violation de la contrainte (2). Dans ce cas, la tâche est donc ordonnancée à b_i^{min} durant l'intervalle $[t_1, t_2[$. Alors la quantité de ressource requise par la tâche i dans $[t_1, t_2[$ est la quantité minimale requise par ces configurations.

Les intervalles $[t_1, t_2[$ sur lesquels appliquer ce test pour le CECSP sont décrits dans [4]. Pour le DECSP, nous devons considérer les projections de ces intervalles sur les entiers, i.e. $[a, b[\rightarrow [\lfloor a \rfloor, \lceil b \rceil[$. Les ajustements pour le CECSP s'adaptent aussi naturellement au DECSP à l'aide de cette même projection.

3 Modèle de programmation linéaire en nombres entiers

3.1 Modèle

La formulation proposée dans cet article est une formulation indexée par le temps. Elle est adaptée de la formulation décrite dans [4]. Dans ces formulations, l'horizon de temps est divisé en intervalles de taille 1 et est défini par : $\mathcal{T}_\mathcal{D}$. Pour chaque activité $i \in \mathcal{A}$ et pour chaque instant $t \in \mathcal{T}_\mathcal{D}$, nous définissons deux variables binaires x_{it} et y_{it} pour modéliser le début et la fin des activités. La variable x_{it} (resp. y_{it}) prendra la valeur 1 si et seulement si l'activité i commence (finit) à l'instant t. Pour modéliser la consommation de ressource et l'apport en énergie, nous introduisons deux variables, b_{it} et w_{it} qui représentent respectivement la quantité de ressource consommée par l'activité i dans la période de temps t et l'énergie reçue par cette même activité durant cette période.

Par manque de place, ce modèle n'est pas entièrement décrit ici mais nous décrivons les contraintes permettant de lier les variables b_{it} et w_{it}, i.e. permettant de calculer l'énergie apportée à i dans la période t, w_{it} , en fonction de la consommation de ressource b_{it}. Nous donnons aussi le nombre de variables et de contraintes du modèle.

Les contraintes liant b_{it} et w_{it}, $\quad \forall t \in \mathcal{T}_\mathcal{D},\ \forall i \in \mathcal{A}$ sont les suivantes :

$$w_{it} = a_i b_{it} + c_i \left(\sum_{\tau=\underline{s_i}}^{t} x_{i\tau} - \sum_{\tau=\underline{s_i}+1}^{t} y_{i\tau} \right) \tag{6}$$

Cette contrainte nous permet de modéliser la fonction de rendement f_i, $\forall i \in \mathcal{A}$. En effet, $\left(\sum_{\tau=\underline{s_i}}^{t} x_{i\tau} - \sum_{\tau=\underline{s_i}+1}^{t} y_{i\tau}\right)$ est égale à 1 si et seulement si l'activité i est en cours à l'instant t. Dans ce cas là, la valeur de l'énergie apportée à i est bien $w_{it} = a_i b_{it} + c_i$. Le second cas se produit quand l'activité i n'est pas en cours à t. Dans ce cas, $b_{it} = 0$ implique $w_{it} = 0$.

Le modèle possède donc $2n|\mathcal{T}_\mathcal{D}|$ variables binaires, $2n|\mathcal{T}_\mathcal{D}|$ variables continues et au plus $3n+|\mathcal{T}_\mathcal{D}|*(6n+1)$ contraintes.

4 Inégalités valides basées sur le raisonnement énergétique

Ce paragraphe décrit des inégalités valides déduites du raisonnement énergétique pour le PLNE. Soit $\mathcal{R}$ l'ensemble des intervalles d'intérêt pour le raisonnement énergétique.

$$(x_{i\underline{s}_i} + y_{i\overline{e}_i} - 1)\,\underline{b}(i, t_1, t_2) + \sum_{j \neq i} \underline{b}(j, t_1, t_2) \leq B(t_2 - t_1) \quad \forall i \in \mathcal{A},\ \forall [t_1, t_2] \in \mathcal{R} \tag{7}$$

$$(x_{i\underline{s}_i} + \sum_{t=t_1}^{t_2} y_{it} - 1)\,\underline{b}(i, t_1, t_2) + \sum_{j \neq i} \underline{b}(j, t_1, t_2) \leq B(t_2 - t_1) \quad \forall i \in \mathcal{A},\ \forall [t_1, t_2] \in \mathcal{R} \tag{8}$$

$$(\sum_{t=t_1}^{t_2} x_{it} + y_{i\overline{e}_i} - 1)\,\underline{b}(i, t_1, t_2) + \sum_{j \neq i} \underline{b}(j, t_1, t_2) \leq B(t_2 - t_1) \quad \forall i \in \mathcal{A},\ \forall [t_1, t_2] \in \mathcal{R} \tag{9}$$

$$(1 - \sum_{t<t_1} x_{it} - \sum_{t>t_2} y_{it})\,\underline{b}(i, t_1, t_2) + \sum_{j \neq i} \underline{b}(j, t_1, t_2) \leq B(t_2 - t_1) \quad \forall i \in \mathcal{A},\ \forall [t_1, t_2] \in \mathcal{R} \tag{10}$$

$$(\sum_{t \leq t_1} x_{it} + \sum_{t \geq t_2} y_{it} - 1)\,\underline{b}(i, t_1, t_2) \leq B(t_2 - t_1) \quad \forall i \in \mathcal{A},\ \forall [t_1, t_2] \in \mathcal{R} \tag{11}$$

L'inégalité (7) correspond au cas où la tâche est centrée et est ordonnancée à b_i^{max} durant $[\underline{s_i}, t_1] \cup [t_2, \overline{e_i}]$. En effet, cette inégalité n'est active que dans le cas où $(x_{i\underline{s}_i} + y_{i\overline{e}_i} - 1) = 1 \Rightarrow \left[x_{i\underline{s}_i} = 1 \wedge y_{i\overline{e}_i} = 1\right]$. Or, ceci implique que la tâche commence à $\underline{s_i}$ et finit $\overline{e_i}$. Donc,

la ressource disponible dans $[t_1, t_2[$ doit être suffisante pour donner la quantité de ressource minimale requise par i dans $[t_1, t_2[$ dans cette configuration. Dans tous les autres cas, l'inégalité devient $\sum_{j \neq i} \underline{b}(j, t_1, t_2) \leq B(t_2 - t_1)$ ou $\sum_{j \neq i} \underline{b}(j, t_1, t_2) - \underline{b}(i, t_1, t_2) \leq B(t_2 - t_1)$.

Les inégalités (8), (9), (10), (11) correspondent respectivement au cas où i est calée à gauche, i est calée à droite, i est complètement incluse dans $[t_1, t_2]$, i est exécutée à b_i^{min} durant l'intervalle $[t_1, t_2]$ et sont déduites de la même façon que (7). Ces inégalités seront ajoutées au modèle indexé par le temps décrit à la section 3 pour renforcer ce dernier.

5 Résultats Expérimentaux

Nous avons testé les différentes méthodes de résolution proposées dans cet article sur les instances de [4]. Les expérimentations ont été conduites sous le système d'exploitation Ubuntu 64-bit 12.04 et les résultats sont calculés au moyen d'un processeur 4-core, 8 thread Core (TM) i7-4770 CPU et de 8GB de mémoire RAM.

Le modèle de PLNE est résolu à l'aide de IBM Cplex 12.6 avec 2 threads et une limite de temps de 100 secondes. Les inégalités déduites du raisonnement énergétique sont calculées avant la résolution du PLNE et ajoutées statiquement au modèle. Ceci augmente la taille du modèle de $5|\mathcal{R}|n$ contraintes (avec $|\mathcal{R}| \in O(n^2)$).

Le tableau 1 décrit les résultats du PLNE.

L'ajout des inégalités du raisonnement énergétique permet de résoudre les instances à 25 tâches de manières plus efficace. Cependant, elles ralentissent le modèle pour les instances à 20 ou 30 tâches mais la perte de rapidité dans ce cas là est beaucoup moins élevée que le gain fait sur les instances à 25. Une poursuite de recherche intéressante serait d'essayer d'ajouter ces contraintes pendant la résolution du PLNE en tant que coupes.

Le modèle de PPC est résolu avec IBM CP Optimizer 12.6. Le tableau 2 décrit les résultats du modèle de PPC. Le modèle de PPC est testé sans ajout du raisonnement énergétique présentés dans cet article mais le modèle utilise les propagateurs du solveur. Des résultats expérimentaux plus détaillées seront proposés lors de la conférence.

Les résultats montrent l'intérêt des inégalités valides ajoutées au PLNE. Le modèle de PPC ne permet pas de prouver l'optimalité des solutions trouvées mais a des résultats similaires au PLNE. En effet, dans presque tous les cas, le modèle de PPC trouve une solution aussi bonne que le PLNE, sans toutefois prouver son optimalité.

Les méthodes présentées ont aussi été testées sur des instances dilatées dans le but de garantir l'existence de solutions entières. La dilatation est effectuée de la manière suivante. Soit α le plus petit commun multiple à tous les b_i^{min} et b_i^{max}. Alors, la dilatation consiste à multiplier $\overline{e}_i$, $\underline{e}_i$, $\overline{s}_i$, $\underline{s}_i$ et W_i par α. Ces expérimentations n'ont pas donné de résultats dû à la grande taille de ces modèles. Les modèles continus pourraient donc rester la seule alternative pour obtenir des solutions optimales dans le cas où la solution est réelle.

Parmi les poursuites de recherche possibles, on trouve l'amélioration des modèles avec la réduction du nombre de variable et/ou de contraintes et la mise en place d'algorithmes de propagation dédiés.

		1^{re} sol.		fin algo.	
	#tâches	temps(s)	écart	temps	%opt.
DEF	20	5.37	7.85	75.4	0.25
ER	20	8.4	10.6	78.9	0.22
DEF	25	4.6	4.4	83.8	0.17
ER	25	0.06	3.86	60.1	0.4
DEF	30	0.99	7.18	75.19	0.25
ER	30	5.66	7.53	75.8	0.25

TABLE 1 – Résultats du PLNE avec et sans inégalités valides : ER et DEF resp. (TL 1000s)

	1^{re} sol.		fin algo.	
#tasks	time	deviation	time lim.	%solved
20	0.19	34.1	100	95
25	0.3	47.9	100	91
30	0.42	43.1	100	95

TABLE 2 – Résultats du modèle PPC

Références

[1] P. Baptiste, C. Le Pape and W. Nuijten (2001). *Constraint-based scheduling*, Kluwer Academic Publishers, Boston/Dordrecht/London.

[2] J. Erschler and P. Lopez (1990). Energy-based approach for task scheduling under time and resources constraints. *2nd International Workshop on Project Management and Scheduling*, pp. 115-121, Compiègne, France.

[3] S. Gay, R. Hartert and P.Schaus (2015). Time-Table Disjunctive Reasoning for the Cumulative Constraint. *CPAIOR 2015, Proceedings*, pp 157–172.

[4] M. Nattaf, C. Artigues, P. Lopez and D. Rivreau (2015). Energetic reasoning and mixed-integer linear programming for scheduling with a continuous resource and linear efficiency functions. *OR Spectrum*, pp 1–34.

[5] P. Vilím (2011). Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources. *CPAIOR 2011, Proceedings*, pp 230–245.

Propagation en Forward Checking pour les contraintes de cardinalité globales imbriquées: Application à un problème de planification de production de papier hygiénique pour réduire les coûts énergétiques

Cyrille Dejemeppe[1] **Olivier Devolder**[2] **Victor Lecomte**[1] **Pierre Schaus**[1]
[1]ICTEAM, UCLouvain, Belgique
[2]N-SIDE, Belgique
{cyrille.dejemeppe, pierre.schaus}@uclouvain.be ode@n-side.com

Résumé

Il est de plus en plus important pour les industries de prévoir une stratégie de réponse aux fluctuations du prix de l'électricité. Cela s'applique aux sites industriels producteurs de papier hygiéniques. Nous proposons un modèle en PPC pour réorganiser le planning de production d'un tel site de telle manière à ce que les dépenses liées au coût de l'électricité soient minimisées. Dans ce modèle, les différentes commandes et contraintes de stock peuvent être encodées à l'aide de p Contraintes Globales de Cardinalité (CGC) ; une pour chaque commande/contrainte de stock. Cependant, cette décomposition propose un filtrage réduit par rapport à celui d'une seule Contrainte Globale de Cardinalité Imbriquée (CGCI)[2]. La complexité temporelle du propagateur de la cohérence d'arc généralisée qu'ils proposent est trop élevée pour s'appliquer dans le cas de grandes instances comme celles des sites industriels susmentionnés. Nous proposons une nouvelle procédure de filtrage en Forward Checking (FWC) pour la CGCI. Nous montrons ensuite les résultats obtenus en appliquant cette procédure à notre problème. Ce document est un résumé de l'article "Forward-Checking Filtering for Nested Cardinality Constraints : Application to an Energy Cost-Aware Production Planning Problem for Tissue Manufacturing" publié à la conférence CPAIOR2016 [1].

1 Réorganisation du planning de production dans l'industrie du papier

Les énergies renouvelables sont de plus en plus présentes dans la production d'électricité des pays de l'union européenne. En conséquence, la volatilité des prix sur le marché européen de l'énergie est de plus en plus importante. Certains sites industriels ont part non négligeable de leurs coûts liée à leur consommation en électricité. Certains sites industriels ont une production offrant de la *flexibilité* : il est possible de moduler leur consommation énergétique. C'est le cas des sites dont le planning de production peut être réorganisé et est flexible ; c'est-à-dire que les processus qui peuvent être déplacés dans le temps ont une consommation énergétique qui diffère d'un processus à l'autre.

Nous considérons le cas d'un site industriel producteur de papier hygiénique. La machine papier produisant des rouleaux de papier a une consommation énergétique qui diffère en fonction du type de papier produit. Lorsqu'un type de papier est produit, il doit être produit pour une période d'une durée minimale pour éviter de calibrer la machine trop souvent. De plus, le passage d'un certain type de papier à un autre se voit attribuer un certain coût puisque le papier produit passera par une phase transitoire pendant laquelle le papier aura une qualité moindre. Le carnet de commande impose qu'un nombre minimal de rouleaux de papier soit produit avant une certaine date. De manière similaire, les contraintes de stockage imposent un nombre maximal de rouleaux de chaque type produit. Nous proposons un modèle en PPC pour ce problème dans lequel les variables représentent les périodes de production et les valeurs associées représentent le type de papier produit à cette période.

2 Nouvelle procédure de propagation FWC pour la CGCI

Le modèle PPC mentionné ci-dessus peut utiliser plusieurs Contraintes Globales de Cardinalité (CGC) pour modéliser le carnet de commandes et les contraintes de stock. La CGC permet de limiter le nombre d'occurrences d'une valeur déterminée dans un ensemble de variables. Il faudra une CGC pour chaque période de temps à laquelle apparaît une commande ou une contrainte de stock. Le filtrage obtenu par une telle décomposition sera cependant inférieur à celui d'un unique propagateur de Contrainte Globale de Cardinalité Imbriquée (CGCI) [2]. Cependant, ce propagateur a une complexité temporelle importante et ne passe pas à l'échelle pour de grandes instances telles que celles liées au problème de la production de papier. Nous proposons donc une procédure de propagation plus légère en Forward Checking (FWC). Cette procédure de propagation en FWC proposée peut-être séparée en deux étapes. La première étape est le calcul de bornes plus étroites renforçant le filtrage. La seconde étape définit un propagateur unifié qui a une complexité amortie réduite en comparaison à une décomposition en plusieurs propagateurs CGC-FWC.

Le calcul de bornes renforcées pour le filtrage se fait en trois étapes :

1. Déduction intra-bornes
2. Déduction inter-bornes
3. Réduction à l'ensemble minimal de bornes

La déduction intra-bornes s'applique pour toutes les variables en ne considérant que les bornes appliquées à une même valeur. Cette déduction s'effectue pour toutes les valeurs, par la succession d'un balayage en avant et en arrière. La déduction inter-bornes considère toutes les bornes présentes pour une variable déterminée. Cette déduction s'applique pour toutes les variables dans la portée de la contrainte. Enfin, après ces deux étapes, il suffit de garder les bornes déduites qui apportent de l'information et d'oublier les autres.

À l'aide des bornes renforcées pré-calculées, il est possible d'obtenir un filtrage plus important en imposant une CGC par variable pour laquelle une borne est présente. Cependant, cela résultera en une complexité temporelle amortie en $\mathcal{O}(p)$ (où p est le nombre de variables pour lesquelles une borne est présente) pour la propagation le long d'une branche. Nous proposons donc un propagateur unifié avec une complexité temporelle réduite en $\mathcal{O}(\log(p))$. Ce propagateur décompose l'ensemble des variables dans la protée de la contrainte en préfixes (p préfixes au total). Ces préfixes tirent avantage d'une structure d'ensembles disjoints et d'un compteur par valeur sur chaque préfixe pour permettre de réduire le temps d'accès et de modification apporté à un ensemble de variables. Ce nouveau propagateur a une complexité temporelle en $\mathcal{O}(\log(p))$.

3 Résultats

Nous avons testé notre modèle de la machine papier en utilisant différents propagateurs pour les contraintes de commande et de stockage. Les résultats montrent que le pré-calcul de bornes renforcées permet un filtrage beaucoup plus important. En effet, un décomposition en multiples CGC-FWC avec les bornes renforcées divise le nombre de backtracks au minimum par deux pour plus de 70% des instances en comparaison avec une décomposition en multiple CGC-FWC sur les bornes originales. De plus, les propagateurs utilisant ces bornes renforcées permettent de résoudre les instances plus rapidement. Par contre, sur ces instances en particulier, notre propagateur unifié ne peut pas être clairement distingué comme plus rapide qu'une décomposition CGC-FWC avec bornes renforcées. Néanmoins, sur d'autre instances plus contraintes, nous pouvons voir un clair gain de temps pour le propagateur unifié.

Enfin, en couplant notre modèle PPC à une approche de Recherche à Large Voisinage (LNS), nous sommes capables de réduire fortement les coûts des plannings de production ; et ce tant au niveau des coûts énergétiques que des coûts associés aux changements de type de papier.

Références

[1] Cyrille Dejemeppe, Olivier Devolder, Victor Lecomte, and Pierre Schaus. Forward-checking filtering for nested cardinality constraints : Application to an energy cost-aware production planning problem for tissue manufacturing. In *Integration of AI and OR Techniques in Constraint Programming*. Springer International Publishing, 2016.

[2] Alessandro Zanarini and Gilles Pesant. Generalizations of the global cardinality constraint for hierarchical resources. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 361–375. Springer, 2007.

La Contrainte Disjonctive avec Temps de Transition

Sascha Van Cauwelaert **Cyrille Dejemeppe** **Pierre Schaus**

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

Résumé

Cet article résume le travail réalisé sur la contrainte disjonctive avec temps de transition [1]. 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. Le travail en question propose un propagateur qui étend les algorithmes de filtrage pour la contrainte disjonctive sans temps de transitions [5]. Les résultats montrent que plusieurs ordres de grandeurs peuvent être gagnés pour une partie non-négligeable d'instances, sans impliquer de surcoût important pour les autres instances.

1 Introduction

Cet article étend les algorithmes de propagation (*Vérification de Surcharge*, *Détection de Précédences*, *Non-Premier/Non-Dernier*, *Découverte d'Arc*)) pour la contrainte disjonctive classique [5] en incluant de la propagation considérant les temps de transitions *séquence-dépendant*. Ils sont utilisés afin de resserrer les bornes d'activités non-préemptives dans des problèmes d'ordonnancement. Ces algorithmes sont tous basés sur le calcul efficace du *plus tôt temps d'achèvement* (*ptta*) d'un ensemble d'activité, en utilisant les structures d'arbre-Θ et d'arbre-Θ-Λ. La contrainte est générique et peut être utilisée pour tout type de problème. L'efficacité de l'approche proposée est démontrée sur des instances du problème d'Ateliers à Cheminement Multiples avec temps de transitions.

2 Contexte

Dans la programmation par contraintes, un problème d'ordonnancement est modélisé en associant trois variables à chaque activité A_i : $debut_i$, fin_i, et $duree_i$ représentant respectivement le temps de début, de fin et de traitement de A_i. Ces variables sont reliées entre elles par la relation suivante :

$$debut_i + duree_i = fin_i$$

Dans le contexte qui nous intéresse, un temps de transition $tt_{i,j}$ est une durée minimum 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 temps de transition impose la relation suivante :

$$\forall i,j : (fin_i + tt_{i,j} \leq debut_j) \vee (fin_j + tt_{j,i} \leq debut_i) \quad (1)$$

3 Extensions pour l'Intégration des Temps de transition

Pour la contrainte disjonctive classique, Vilim a construit plusieurs algorithmes de propagation qui reposent tous sur un calcul efficace du ptta d'un ensemble d'activités Ω (écrit $ptta_\Omega$) à l'aide de structures de données appelées arbre-Θ et arbre-Θ-Λ [5]. Notre contribution est un calcul plus resserré de la borne inférieure $ptta_\Omega$ en tenant compte des temps de transition entre les activités au sein de ces structures.

Un des objectifs de notre travail était de conserver la complexité originale des algorithmes de Vilim, $\mathcal{O}(n.log(n))$. Sans rentrer dans les détails, nous nous basons uniquement sur la cardinalité de $\Theta \subseteq \Omega$, et pré-calculons une borne inférieure des temps de transition au sein de cet ensemble, que nous écrivons $\underline{tt}(k)$, k étant ici la cardinalité d'un Θ donné. Ce pré-calcul est basé sur des relaxations du *problème du plus court chemin sous contraintes*, ce dernier étant NP-Difficile. Nous utilisons la meilleure borne obtenue à partir d'une *forêt de poids minimum*, d'un *programme dynamique*, d'un *flux de coût minimum* et d'une *relaxation*

lagrangienne. Les noeuds de l'arbre-Θ (et de l'arbre-Θ-Λ) sont alors étendus à l'aide d'une simple information : la cardinalité de l'ensemble d'activités représenté par ce noeud. Les mises à jour de $ptta_\Omega$ ainsi que les algorithmes de filtrage sont adaptés en fonction.

4 Évaluation

Pour évaluer notre contrainte, nous avons utilisé le solveur OscaR[2] et des processeurs AMD Opteron (2,7 GHz). L'évaluation porte sur les 960 instances [1] introduites dans [1]. Pour chaque instance considérée, nous avons utilisé les 3 filtrages suivants pour la contrainte disjonctive avec temps de transition :

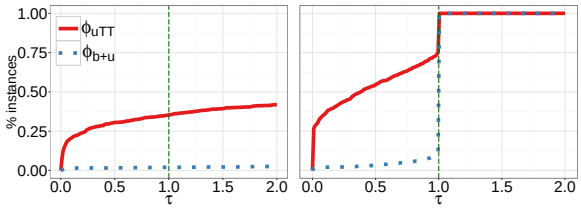
1. Les contraintes binaires [2] (ϕ_b) données dans l'Équation 1.
2. Les contraintes binaires de l'Équation 1 avec la contrainte disjonctive de [5] (ϕ_{b+u}).
3. La contrainte introduite dans cet article (ϕ_{uTT}).

Afin de présenter des résultats justes ne mesurant que les avantages dus à notre contrainte, nous avons suivi la méthodologie introduite dans [4, 3]. En bref, cette évaluation propose de pré-calculer un arbre de recherche en utilisant le filtrage qui élague le moins, pour ensuite le *rejouer* en utilisant les différents filtrages étudiés. La génération se base ici sur la stratégie de recherche Set Times et son temps est limité à 600 secondes. Nous avons construit des *profils de performance* comme décrit dans [4, 3]. Fondamentalement, ce sont des fonctions de distribution cumulative d'une métrique de performance τ.

Les Figures 1a et 1b fournissent respectivement les profils pour le temps et le nombre de retours arrières pour toutes les instances. On peut voir que ϕ_{b+u} n'apporte quasiment aucun avantage, d'un point de vue du temps et du nombre de retours arrières. Par contre, dans $\sim 20\%$ des cas, ϕ_{uTT} est environ 10 fois plus rapide que ϕ_b, et nous résolvons $\sim 35\%$ des instances plus rapidement. En outre, il offre plus d'élagage dans $\sim 75\%$ des cas. Plus de détails et d'interprétations sont disponibles dans [1].

5 Conclusion

Dans cet article, nous avons proposé d'étendre les algorithmes de propagation de la contrainte disjonctive classique afin qu'ils considèrent les temps de transition. Nous avons étendu les structures d'arbre-Θ et d'arbre-Θ-Λ en y intégrant une borne inférieure quant aux temps de transition requis par un nombre donné d'activités. Ceci permet de renforcer le calcul du *plus tôt temps d'achèvement* d'un ensemble d'activités, renforçant ainsi les inférences réalisées par les algorithmes de filtrages. Nous avons démontré que l'élagage supplémentaire obtenu par notre contrainte peut réduire considérablement le nombre des noeuds (et le temps nécessaire pour résoudre le problème) sur une large portions des instances considérées.



(a) τ : rapport de temps. (b) τ : rapport de retours arrière.

FIGURE 1 – Profils de performance pour les 960 instances.

Références

[1] 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.

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

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

[4] 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.

[5] 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. Disponibles via l'URL : `http://becool.info.ucl.ac.be/resources/benchmarks-unary-resource-transition-times`.

2. Pour des raisons d'efficacité, des propagateurs dédiés ont été mis en oeuvre au lieu d'imposer des contraintes réifiées.

De la résilience de la propagation unitaire aux transformations par max-résolution

André Abramé **Djamal Habet**

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

Résumé

Les solveurs de type séparation et évaluation pour Max-SAT utilisent la propagation unitaire et la règle de la max-résolution lors du calcul de la borne inférieure pour détecter et transformer les sous-ensembles inconsistants (SI). Nous montrons que les transformations par max-résolution affectent la capacité de la propagation unitaire à détecter des SI. Nous introduisons la notion d'UP-résilience qui permet de caractériser les transformations n'affectant pas le fonctionnement de la propagation unitaire et plus généralement de quantifier cet impact. Nous discutons des propriétés liés à ces transformations et présentons des résultats expérimentaux qui valident la pertinence de notre analyse. Ce travail a été publié à la conférence IJCAI 2015 [2].

1 Introduction

La propagation unitaire (*unit propagation*, UP) est une règle d'inférence sémantique qui consiste à satisfaire itérativement les littéraux apparaissant dans des clauses unitaires jusqu'à ce qu'un conflit soit détecté (i.e. une clause vide) ou qu'il n'y ait plus aucune clause unitaire. Lorsqu'une clause vide est détectée, l'ensemble des clauses qui ont menées, par propagation unitaire, à falsifier ses littéraux forment un sous-ensemble inconsistant de la formule.

Cette méthode est utilisée par les solveurs de type séparation et évaluation lors du calcul de la borne inférieure pour détecter les sous-ensembles inconsistants. De la capacité de la propagation unitaire à détecter les sous-ensembles inconsistants dépend la précision de l'estimation de la borne inférieure, et donc la capacité des solveurs à faire des coupes dans l'arbre de recherche.

Une fois détectés, les sous-ensembles inconsistants doivent être transformés pour s'assurer qu'ils ne soient comptés qu'une fois. Une des méthodes de transformation existantes est basée sur la règle de la max-résolution, qui consiste à remplacer les clauses des sous-ensembles inconsistants par une clause résolvante et un ensemble de clauses de compensation.

2 UP-résilience

Après la transformation d'un sous-ensemble inconsistant par max-résolution, les informations utilisables par la propagation unitaire dans la formule originale peuvent être divisées (ou "fragmentées") dans plusieurs clauses de compensation. Elles ne sont donc pas directement utilisables par la propagation unitaire et une ou plusieurs étapes de max-résolution sont nécessaires pour les rendre exploitables. Quand de telles transformations sont conservées, cela peut affecter la capacité de la propagation unitaire à détecter des sous-ensembles inconsistants dans les autres nœuds de l'arbre de recherche. Par conséquent, l'estimation de la borne inférieure peut être moins précise et les solveurs peuvent être conduits à explorer davantage de nœuds.

Nous avons donc introduit la notion d'*UP-résilience* des transformations qui caractérise les transformations qui n'affectent pas l'efficacité de la propagation unitaire.

Dans un graphe d'implications Quand le phénomène de fragmentation se produit, les clauses de compensation qui auraient pu propager un littéral l apparaissant dans un graphe d'implications G contiennent d'autres littéraux qui ne sont pas des voisins de l dans G. Par conséquent, pour détecter si une transformation n'est pas affectée par ce phénomène de fragmentation il est possible de vérifier si les littéraux de G

peuvent être propagés lorsque seuls leurs voisins dans G sont satisfaits. Si c'est le cas, nous dirons que la transformation est UP-résiliente dans G.

Généralisation Cette première définition de l'UP-résilience dépend du voisinage des littéraux dans le graphe d'implications. Cependant, un même sous-ensemble inconsistant peut être détecté par plusieurs séquences de propagations pouvant être décrites par plusieurs graphes d'implications distincts. Nous avons donc généralisé la précédente définition, en considérant tout les graphes d'implications pouvant mener à la détection d'un sous-ensemble inconsistant. Nous dirons qu'une transformation est UP-résiliente ssi pour tout voisinage v de tout littéral l apparaissant dans le sous-ensemble inconsistant, l peut être propagé dans la formule transformée lorsque v est satisfait par l'interprétation courante.

Nous avons également introduit la notion de pourcentage d'UP-résilience d'une transformation, mesuré comme le pourcentage de couples (voisinage, littéral) respectant la définition précédente.

Propriétés Une des propriétés les plus intéressantes des transformations UP-résilientes est la capacité de "récupérer" les propagations qui ne sont plus nécessaires aux sous-ensembles inconsistants transformés. Nous avons ainsi montré que si une transformation est UP-résiliente pour un ensemble de littéraux, on a les mêmes propriétés pour l'ensemble des littéraux que pour chacun d'entre eux individuellement.

Nous avons également montré que les transformations apprises par les schémas d'apprentissage utilisés par les solveurs de type séparation et évaluation de l'état de l'art était UP-résiliente. Cela contribue à expliquer d'un point de vue théorique leur efficacité, qui n'avait été démontrée qu'empiriquement jusqu'alors.

3 Étude expérimentale

Nous avons implémenté dans AHMAXSAT [1] une procédure simple permettant de calculer le pourcentage d'UP-résilience des transformations. Nous avons évalué certains composants du solveur au regard de l'UP-résilience avant de chercher à valider la pertinence de la notion d'UP-résilience elle-même. Les résultats obtenus (non détaillés ici), montrent que :

- l'ordre d'application des étapes de max-résolution a un impact important sur l'UP-résilience des transformations et donc sur les performances des solveurs ;
- les schémas d'apprentissage utilisés actuellement ne permettent pas de détecter toutes les transformations UP-résiliente et donc que les schémas d'apprentissage peuvent être améliorés ;
- l'UP-résilience permet de quantifier de manière précise l'impact des transformations sur l'efficacité de la propagation unitaire.

4 Conclusions

Nous avons introduit la notion d'UP-résilience des transformations, qui permet de quantifier l'impact de la max-résolution sur le mécanisme de la propagation unitaire. Nous avons montré que, selon le critère de l'UP-résilience, les motifs utilisés dans les schémas d'apprentissage des solveurs de type séparation et évaluation existant n'affectent pas l'efficacité de la propagation unitaire. Ceci contribue à expliquer d'un point de vue théorique l'efficacité des ces schémas d'apprentissage qui avait été démontré uniquement de manière empirique jusqu'alors. Les résultats de l'étude expérimentale que nous avons menée montrent la pertinence de la notion d'UP-résilience. Ils mettent également en évidence les forces et les lacunes de certains des composants liés à l'application de la max-résolution et à l'apprentissage.

Ces résultats ouvrent de nouvelles perspectives. Parmi elles, il serait intéressant de développer de nouveaux ordres d'application de la max-résolution visant à accroître le pourcentage d'UP-résilience des transformations. Les schémas d'apprentissage existants peuvent également être améliorés, soit en établissant une caractérisation des transformations UP-résilientes qui puisse être vérifiée efficacement ou en complétant les ensembles de motifs existants. Maintenant que les effets des transformations par max-résolution sont mieux compris, on peut envisager d'appliquer l'apprentissage non seulement vers le bas de l'arbre de recherche, comme c'est actuellement le cas, mais également vers le haut. Les bénéfices escomptés seraient de réduire encore davantage la redondance dans le calcul de la borne inférieure et de permettre d'exploiter certaines propriétés structurelles des instances.

Références

[1] André Abramé and Djamal Habet. AHMAXSAT : Description and evaluation of a branch and bound Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 9 :89–128, 2015.

[2] André Abramé and Djamal Habet. On the resiliency of unit propagation to max-resolution. In *IJCAI 2015*, pages 268–274. 2015.

Un *back-end* d'optimisation continue stochastique pour MiniZinc avec des applications à des problèmes de placements géométriques

Thierry Martinez[1] **François Fages**[1] **Abder Aggoun**[2]

[1] Inria, Team Lifeware, France
[2] KLS-Optim, France
{thierry.martinez,francois.fages}@inria.fr abder.aggoun@klsoptim.com

Résumé

Ce papier est un résumé de l'article publié à CPAIOR'16 : « A Stochastic Continuous Optimization Backend for MiniZinc with Applications to Geometrical Placement Problems » *[3].*

MiniZinc est un langage de modélisation par contraintes indépendant du solveur qui est de plus en plus utilisé dans la communauté de la programmation par contraintes. Il peut être utilisé pour comparer différents solveurs, qui peuvent reposer sur la programmation par contraintes, la satisfiabilité booléenne (SAT), la programmation linéaire mixte en nombres entiers (*Mixed Integer Linear Programming*, MILP), et plus récemment la recherche locale (*Local Search*). Dans ce papier, nous présentons un solveur d'optimisation continue stochastique pour les modèles MiniZinc sur les nombres réels. Plus précisément, nous décrivons la traduction de modèles FlatZinc vers des fonctions-objectifs sur les réels, et leur utilisation dans le solveur CMA-ES, la stratégie évolutionnaire adaptative par matrice de covariance (Covariance Matrix Adaptation Evolution Strategy). Nous illustrons cette approche en modélisant déclarativement et en résolvant des problèmes difficiles de placements géométriques, motivés par des applications en logistique impliquant des formes mixtes carrées et courbes et des formes complexes définies par des courbes de Bézier.

1 Introduction

MiniZinc [5] est un langage de modélisation par contraintes de « moyen niveau » qui est en train de devenir un standard dans la communauté de programmation par contraintes. Ce langage est de suffisamment haut niveau pour exprimer aisément la plupart des problèmes de contraintes, mais de suffisamment bas niveau pour être traduit facilement et de façon cohérente dans les formats d'entrée attendus par les solveurs existants. Cette traduction de MiniZinc vers les formats d'entrée spécifiques aux solveurs passe par un applatissement qui prend en entrée une instance MiniZinc et qui produit une instance FlatZinc. FlatZinc est un langage d'entrée de bas niveau conçu pour être facile à traduire vers chaque solveur.

Jusqu'à présent, il existe des supports FlatZinc pour la programmation linéaire mixte en nombres entiers (*Mixed Integer Linear Programming*, MILP, comme CPLEX), les solveurs de programmation par constraintes en domaine fini (Choco [1], Gecode [2], SICStus prolog, ...), les solveurs SAT (MinisatID, ...) et récemment la recherche locale (iZplus [3], Oscar-cbls [1]).

Dans ce papier, nous étudions un solveur d'un autre genre basé sur l'optimisation stochastique continue pour résoudre des instances FlatZinc sur les nombres réels, à savoir la stratégie *évolutionnaire adaptative par matrice de covariance* (CMA-ES, *Covariance Matrix Adaptation Evolution Strategy*) [2]. Le choix de CMA-ES parmi les autres algorithmes évolutionnaires ou d'optimisation d'essaims de particules est motivé par l'absence de paramétrage pour cet algorithme et par ses performances sur les problèmes difficiles.

Nous illustrons notre *back-end* CMA-ES pour FlatZinc en l'appliquant à la résolution de problèmes de placements géométriques difficiles qui, à notre connaissance, vont au-delà de l'état de l'art de la modélisation déclarative et de la résolution avec contraintes.

1. https://github.com/chocoteam/choco-parsers
2. http://www.gecode.org/flatzinc.html
3. http://www.minizinc.org/challenge2014/descriptionizplus.txt

2 Travaux liés

La plupart des *back-ends*, des implémentations de solveurs pour FlatZinc, sont ainsi dédiées aux domaines discrets. Dans [1], Björdal et al. ont présenté un *back-end* de recherche locale basé sur des contraintes pour MiniZinc.

Nous montrions déjà dans [4] que les contraintes de non-chevauchement entre formes géométriques peuvent être associés à une *mesure de chevauchement* entre les objets qui peut être directement utilisée dans une fonction de coût de CMA-ES. Dans [6], Salas et Chabert ont montré que les mesures qui étaient définies de façon *ad hoc* dans [4] pouvaient être calculées par des méthodes d'intervalle dans IBEX [4] avec un algorithme numérique qui mesure automatiquement la *distance de pénétration*.

3 Résultats

L'extrait de code ci-dessous définit un prédicat MiniZinc pour la contrainte de non-chevauchement entre deux cercles $((x_1, y_1), r_1)$ et $((x_2, y_2), r_2)$ décrits par les coordonnées de leur centre et leur rayon, en calculant la distance euclidienne des centres.

```
predicate non_overlap_circles(
  var float: x1, var float: y1, var float: r1,
  var float: x2, var float: y2, var float: r2) =
  pow(x1-x2, 2) + pow(y1-y2, 2) > pow(r1+r2, 2);
```

On obtient une mesure de coût associée en calculant un degré de violation pour cette contrainte. Pour une telle inégalité, il s'agit de l'excédent de la différence entre le membre droit et le membre gauche : $\max((r_1 + r_2)^2 - (x_1 + x_2)^2 - (y_1 + y_2)^2, 0))$. Le *back-end* génère une fonction C calculant et agrégeant ces coûts.

En appelant cette fonction dans la boucle d'optimisation de CMA-ES, on peut obtenir des placements comme celui à gauche de la figure 1, avec 18 cercles de rayon $i^{-1/2}$ pour $1 \leq i \leq 18$.

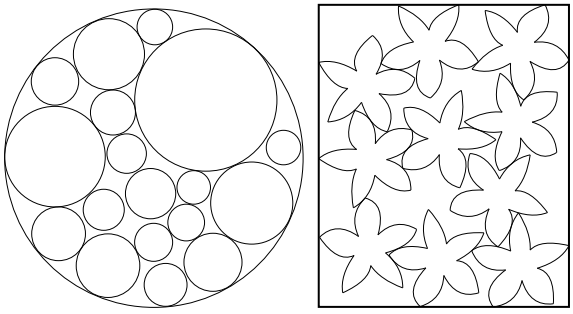


FIGURE 1 – Placement de 18 cercles de rayon $i^{-1/2}$ (gauche) et de 10 rosaces de Bézier (droite).

4. `http://www.ibex-lib.org`

Le placement à droite de la figure 1 a été obtenu pour 10 rosaces dont les contours sont décrits par des courbes de Bézier : dans l'article, nous décrivons une mesure de chevauchement approchée pour les courbes de Bézier dérivée du calcul des intersections.

4 Conclusion

Nous avons donné des définitions générales en MiniZinc pour les distances de pénétration, ou des mesures de chevauchement plus simples, entre les polygones, les cercles, et des formes complexes définies par des courbes de Bézier, motivés par des applications dans l'industrie cosmétique et automobile. Les résultats obtenus montrent la performance de MiniZinc-CMAES à la fois en termes de déclarativité de la modélisation et de résolution efficace (quoique suboptimale) de problèmes de placements géométriques très difficiles avec des formes complexes et des rotations continues.

Références

[1] Gustav Björdal, Jean-Noël Monette, Pierre Flener, and Justin Pearson. A constraint-based local search backend for minizinc. *Constraints*, 20(3) :325–345, 2015.

[2] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2) :159–195, 2001.

[3] Thierry Martinez, François Fages, and Abder Aggoun. A stochastic continuous optimization backend for minizinc with applications to geometrical placement problems. In *Proceedings of the 13th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming, CPAIOR'16*, May 2016.

[4] Thierry Martinez, Lumadaiara Vitorino, François Fages, and Abderrahmane Aggoun. On solving mixed shapes packing problems by continuous optimization with the cma evolution strategy. In *Proceedings of the first Computational Intelligence BRICS Congress BRICS-CCI'13*, pages 515–521. IEEE Press, September 2013.

[5] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc : Towards a standard CP modelling language. In *CP*, pages 529–543, 2007.

[6] Ignacio Salas and Gilles Chabert. Packing curved objects. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI'15*, Buenos Aires, Argentina, 2015.

Algorithmes de filtrage pour la contrainte WeightedCircuit

Sylvain Ducomman[1] [2] * **Hadrien Cambazard**[1] **Bernard Penz**[1]

[1] Univ. Grenoble Alpes, G-SCOP, F-38000 Grenoble, France
CNRS, G-SCOP, F-38000 Grenoble, France
[2] Geoconcept SA, 92220 Bagneux, France
{prénom.nom}@grenoble-inp.fr sylvain.ducomman@geoconcept.com

Résumé

La présence de circuit dans un graphe pondéré est une contrainte essentielle de beaucoup de problèmes, notamment pour les problèmes de tournées de véhicules. La contrainte WeightedCircuit, permettant de maintenir un circuit hamiltonien dans de tels graphes, est donc une contrainte clef dans ce type de problèmes. Notre étude *Alternative Filtering for the Weighted Circuit Constraint : Comparing Lower Bounds for the TSP and Solving TSPTW* présentée à *AAAI-16* propose des algorithmes de filtrage pour la contrainte WeightedCircuit basés sur des relaxations du Problème du Voyageur de Commerce. Nous proposons aussi une étude théorique des bornes inférieures fournies par les relaxations employées.

1 Introduction

La contrainte WeightedCircuit permet de maintenir un circuit hamiltonien dans un graphe pondéré représentant souvent la distance sur chaque arc du graphe. Dans cette étude [3], nous proposons différents algorithmes de filtrage pour la contrainte WeightedCircuit. Cette dernière est utilisée dans un modèle de Programmation par Contraintes pour le Problème (PPC) du voyageur de commerce avec fenêtres de temps (TSPTW).

On considère un graphe orienté complet $G(N, E)$ dans lequel E est l'ensemble des arcs et $N = \{0, \ldots, n+1\}$ est l'ensemble des noeuds. $d_{i,j}$ (resp. $t_{i,j}$) représente la distance (resp. le temps) entre les noeuds i et j. La modélisation en PPC du TSPTW est basée sur trois variables. Les variables $next_i \in N$ représentent le successeur direct du noeud i dans la tournée. Les deux autres variables utilisées sont des variables de cumul de quantités, la première pour la distance cumulée $dist$ et la seconde pour le temps de trajet parcouru $start$ ($dist_0 = 0$ et $start_0 = 0$). Nous améliorons ce modèle en explicitant les relations de précédences entre les noeuds du graphe grâce entre autres aux informations des fenêtres de temps. Ainsi, des variables redondantes sont ajoutées au modèle : des variables booléennes b, indiquant si un noeud i doit se trouver avant un noeud j, et des variables pos, représentant la position d'un noeud dans la tournée. Les contraintes redondantes permettent de lier les variables additionnelles aux variables existantes. Ci-dessous un modèle simplifié de [3] :

Minimize z

$$z = \sum_{i=0}^{n} (d_{i,next_i})$$

$\text{WeightedCircuit}(next_0, \ldots, next_{n+1}, z)$

$$dist_{next_i} = dist_i + d_{i,next_i} \qquad \forall i \in N \backslash \{n+1\}$$

$$start_{next_i} \geq start_i + t_{i,next_i} \qquad \forall i \in N \backslash \{n+1\}$$

$$b_{ij} + b_{ji} = 1$$

$$pos_i = \sum_{j \in N} b_{ji} \qquad \forall i \in N$$

$\text{AllDifferent}(pos_0, \ldots, pos_{n+1})$

$$(b_{ij} = 1) \Rightarrow next_j \neq i \qquad \forall (i,j) \in E$$

$$pos_j > pos_i + 1 \Rightarrow next_i \neq j \qquad \forall (i,j) \in E$$

$$pos_j > pos_i \Leftrightarrow b_{ij} = 1 \qquad \forall (i,j) \in E$$

$$(b_{ij} = 1) \Rightarrow start_j \geq start_i + t_{ij} \qquad \forall (i,j) \in E$$

$$start_i + t_{ij} > start_j \Rightarrow next_i \neq j \qquad \forall (i,j) \in E$$

*Papier doctorant : Sylvain Ducomman[1] [2] est auteur principal.

2 WeightedCircuit

L'existence d'un circuit hamiltonien dans un graphe est un problème NP-complet. Pour effectuer le filtrage de cette contrainte nous utilisons des algorithmes de filtrage basés sur les coûts. Ces derniers s'appuient sur des relaxations connues du Problème du Voyageurs de Commerce. Ainsi, trois relaxations sont présentées. La première est basée sur le problème d'arbre couvrant de poids minimum, présentée par [5] et dont l'algorithme de filtrage est utilisé dans [1]. La seconde relaxation repose une affectation de poids minimum. Nous étendons l'algorithme de filtrage utilisé dans [4] en calculant les coûts réduits exacts du problème, présentés dans [6]. Enfin, la dernière relaxation présentée s'appuie sur un plus court chemin de n arcs [2]. Cette dernière a l'avantage de pouvoir prendre en compte les positions des noeuds dans le programme dynamique utilisé. Ainsi, l'algorithme de filtrage permet de filtrer sur les variables *next* d'un noeud dans le graphe mais aussi sur les variables *pos*. Cette relaxation se révèle efficace lorsque des contraintes auxiliaires viennent s'ajouter au problème, comme les contraintes de fenêtres de temps par exemple. Les relaxations basées sur l'arbre couvrant de poids minimum et celles sur le plus court chemin de n arcs peuvent être améliorées par relaxation lagrangienne, ce qui permet d'une part de fournir de meilleures bornes inférieures et d'autre part d'effectuer le filtrage à différentes étapes du processus de sous-gradient. Chaque algorithme de filtrage pour une relaxation dédiée est idempotent. Cependant dans leurs versions lagrangiennes, l'idempotence n'est plus vérifiée.

3 Bornes inférieures

Une partie de ces travaux est consacrée à l'étude des bornes inférieures fournies par les différentes relaxations utilisées. Ainsi, nous avons prouvé théoriquement que les relaxations utilisées ne sont pas comparables. C'est-à-dire qu'aucune d'entre elle est dominante. Ce résultat est montré pour le cas général des relaxations mais aussi pour les versions lagrangiennes. Il peut donc être intéressant de combiner les algorithmes de filtrage.

4 Résultats

Les résultats principaux de l'étude comparent les différents algorithmes de filtrage pour différents benchmarks du TSP et du TSPTW. Les résultats sur les benchmarks du TSP montrent que l'algorithme de filtrage performant reste celui de la littérature utilisant les arbres couvrants de poids minimum. En revanche, pour les benchmarks du TSPTW, les nouveaux algorithmes de filtrages profitent davantage des informations des positions liées aux fenêtres de temps. En effet, nous montrons que dans la plupart des cas, l'algorithme de filtrage utilisant le problème de plus court chemin donne de meilleures bornes inférieures ainsi qu'un meilleur filtrage des variables *next* et *pos* au noeud racine. De plus, cet algorithme est aussi performant pour la résolution des différents benchmarks pour le TSPTW. Les nouvelles bornes proposées tirent donc un meilleur parti de l'information supplémentaire disponible sur les positions.

5 Conclusion

L'étude [3] propose trois filtrages différents pour la contrainte WeightedCircuit. Cette dernière est utilisée dans un modèle de Programmation par Contraintes pour résoudre le problème du voyageur de commerce avec fenêtres de temps. Les résultats obtenus montrent que pour ce type de problèmes, le meilleur algorithme de filtrage utilise une relaxation permettant de raisonner sur les positions des noeuds. Dans de futurs travaux, il sera intéressant d'évaluer si les relaxations peuvent se combiner avantageusement en pratique.

Références

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

[2] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2) :145–164, 1981.

[3] Sylvain Ducomman, Hadrien Cambazard, and Bernard Penz. Alternative filtering for the weighted circuit constraint : Comparing lower bounds for the TSP and solving TSPTW. In *AAAI-16*, Phoenix, Arizona, 2016.

[4] Filippo Focacci, Andrea Lodi, and Michela Milano. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing*, 14(4) :403–417, 2002.

[5] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6) :1138–1162, 1970.

[6] Jean-Charles Régin. Cost-based arc consistency for global cardinality constraints. *Constraints*, 7(3-4) :387–405, 2002.

Un Filtrage Simple et Echelonnable de la règle Time-Table pour la Contrainte Cumulative

Steven Gay, Renaud Hartert, Pierre Schaus

UCLouvain, ICTEAM,
Place Sainte Barbe 2,
1348 Louvain-la-Neuve, Belgium
{prenom.nom}@uclouvain.be

Résumé

La contrainte Cumulative est une des contraintes centrale de la programmation par contrainte. Nous montrons deux algorithmes pour la règle de filtrage Time-Table : un théorique en O(n), et un pratique conçu pour passer à l'échelle en utilisant de l'incrémentalité.

1 Introduction

La contrainte **Cumulative** est essentielle dans le cadre PPC, et se retrouve dans les applications d'ordonnancement et de placement. La règle de filtrage la plus basique est le Time-Tabling. Les algorithmes de Time-Tabling ont été revus et améliorés plusieurs fois ces dernières années. La meilleure complexité théorique connue est $O(n \log n)$, par Ouellet et Quimper.

1.1 Définition de la contrainte Cumulative

La contrainte **Cumulative** modélise une ressource de capacité $C \in \mathbb{N}$ qui contient un ensemble de tâches Ω. Une tâche est un objet défini par plusieurs variables entières : un début $s \in \mathbb{Z}$ (start), une fin $e \in \mathbb{Z}$ (end), une durée $d \in \mathbb{N}$ (duration) et une hauteur $h \in \mathbb{N}$ (height). Nous supposerons que la durée et la hauteur des tâches sont fixées, seuls le début et la fin sont des variables de décision ; de plus le début et la fin sont reliés par la durée et doivent vérifier $s + d = e$, de sorte qu'il n'y a vraiment qu'une variable de décision par tâche.

La contrainte **Cumulative** maintient la relation

$$\forall t \in \mathbb{Z} : \sum_{i \in \Omega \,:\, s_i \leq t < e_i} c_i \leq C$$

Autrement dit, au temps t, la somme des hauteurs des tâches traitées à t ne peut dépasser la capacité.

1.2 Time-Tabling

Le Time-Tabling confronte le profil obligatoire à chaque tâche pour filtrer les bornes de celles-ci. Le profil obligatoire d'un ensemble de tâches Ω est l'aggrégation de leurs parties obligatoires :

$$TT_\Omega = t \in \mathbb{Z} \longrightarrow \sum_{i \in \Omega \,|\, \overline{s}_i \leq t < \underline{e}_i} c_i$$

La règle qui filtre les bornes gauches des tâches s'assure que le support gauche est compatible avec le profil :

$$(t < \underline{e}_i) \;\wedge\; (c_i + TT_{\Omega \setminus i}(t) > C) \;\Rightarrow\; t < s_i$$

La règle de filtrage à droite est symétrique.

2 État de l'art

Différents algorithmes ont été créés pour effectuer le Time-Tabling. Aucun n'est idempotent, et chacun peut donc demander plusieurs itérations. Par exemple, le filtrage d'une borne inférieure peut créer ou agrandir une partie obligatoire, ce qui peut demander à un algorithme d'être redémarré pour prendre en compte la nouvelle partie obligatoire. Une autre cause de non-idempotence est l'effet de ping-pong entre le filtrage des supports gauche et celui des supports de droite.

L'algorithme de balayage de [1] filtre les bornes des tâches et construit le profil courant en même temps. Il a longtemps été le standard. L'algorithme de [5] est un balayage synchronisé, qui filtre et construit le profil courant en prenant en compte le filtrage de la passe

courante. Il demande moins d'itérations, une éventuelle nouvelle partie obligatoire ne le forçant pas à redémarrer, et a été conçu pour passer à l'échelle. Malheureusement cet algorithme n'est pas trivial à implémenter. L'algorithme basé sur des arbres d'intervalles interdits de [6] demande plus d'itérations que le balayage, mais arrive à une complexité de $O(n \log n)$.

3 Filtrage Time-Tabling en temps linéaire

Nous exposons un nouvel algorithme en $O(n)$, qui se repose sur un algorithme de requête de minimum sur un intervalle (range minimum query, RMQ). Après un prétraitement linéaire, le RMQ de [2] est capable de répondre à des requêtes d'élément minimum du tableau dans un intervalle d'indices en $O(1)$.

Pour obtenir un Time-Tabling linéaire, il suffit de commencer par écrire les hauteurs du profil obligatoire dans un tableau. Ensuite, on peut localiser l'indice de début et l'indice de fin de tous les supports en temps linéaire, par exemple en triant les débuts et fin de supports avec les temps de changement du profil obligatoire dans un même tableau.

On se sert alors du RMQ dans sa version maximum pour déterminer en $O(1)$ par tâche la hauteur maximum du profil entre le début et la fin d'un support, ce qui permet de savoir si un support violerait la capacité, et de filtrer les valeurs fautives le cas échéant.

Cette approche est d'un intérêt foncièrement plus théorique que pratique, à cause du surcoût en nombre d'itérations et des constantes impliquées dans le RMQ.

4 Filtrage Time-Tabling échelonnable

Nous présentons ensuite un algorithme pratique, qui se comporte mieux que sa complexité théorique en $O(n^2)$.

Dans l'article [3], nous présentons une alternative simple basée sur le filtrage de [1]. L'algorithme commence par construire le profil obligatoire, en vérifiant que celui-ci ne viole pas la capacité. Construire le profil demande de trier les tâches qui ont une partie obligatoire : ceci est fait en gardant les résidus ([4]) des derniers tris, ce qui donne un coût en pratique linéaire. Ensuite, il localise les tâches sur ce profil en utilisant les résidus de localisation de la dernière passe, puis en se tournant vers une recherche binaire en $O(\log n)$ par défaut. Une fois la tâche localisée, on cherche à valider le support courant où à filtrer les valeurs de la même manière que [1], mais avec seulement cette tâche, ce qui simplifie les structures de données à utiliser pendant le balayage. L'algorithme utilise le même profil pour filtrer les bornes gauche et droite. Finalement, il écarte incrémentalement une partie des tâches qui ne peuvent pas participer au filtrage, ce qui réduit en général considérablement le travail à effectuer dans les phases précédentes.

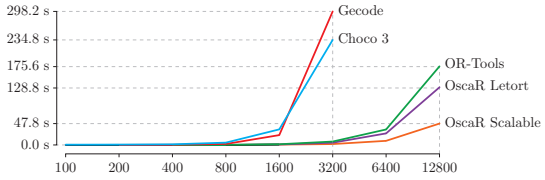


FIGURE 1 – Comparaison d'algorithmes de Time-Tabling : sur des instances de taille 100 à 12800, temps de descente sur une branche en utilisant une heuristique min-min.

Cet algorithme est testé avec les Time-Tablings implémentés concrètement dans d'autres solveurs PPC et notre implémentation de [5] [1]. Les résultats montrent que notre algorithme échelonnable surpasse les autres sur des grandes instances de problèmes d'ordonnancement, parfois de manière significative.

Références

[1] Nicolas Beldiceanu and Mats Carlsson. A new multi-resource cumulatives constraint with negative heights. In *CP*, Lecture Notes in Computer Science, pages 63–79, 2002.

[2] Johannes Fischer and Volker Heun. Theoretical and practical improvements on the rmq-problem, with applications to lca and lce. In *Combinatorial Pattern Matching*, pages 36–48. Springer, 2006.

[3] Steven Gay, Renaud Hartert, and Pierre Schaus. Simple and scalable time-table filtering for the cumulative constraint. In *Principles and Practice of Constraint Programming*, pages 149–157. Springer, 2015.

[4] Christophe Lecoutre, Fred Hemery, et al. A study of residual supports in arc consistency. In *IJCAI*, volume 7, pages 125–130, 2007.

[5] Arnaud Letort, Nicolas Beldiceanu, and Mats Carlsson. A scalable sweep algorithm for the cumulative constraint. In *Principles and Practice of Constraint Programming*, pages 439–454. Springer, 2012.

[6] Pierre Ouellet and Claude-Guy Quimper. Time-table extended-edge-finding for the cumulative constraint. In *Principles and Practice of Constraint Programming*, pages 562–577. Springer, 2013.

1. Le code et les instances sont disponibles sur http://bit.ly/cumulativett

Une heuristique de recherche par ordre de conflits pour des problèmes d'ordonnancement

Steven Gay[1], **Renaud Hartert**[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

Les heuristiques de recherche adaptatives sont devenues un outil essentiel en programmation par contrainte. Nous présentons une variante de Last Conflict qui prend la main sur l'heuristique de recherche plutôt que de la réparer, en minimisant récursivement les ensembles de variables en conflit. Cet article est un résumé de [1].

1 Introduction

Le backtracking est un algorithme central pour la résolution de problèmes combinatoires. Malheureusement il a tendance à visiter fréquemment les mêmes sous-arbres sans solutions, un phénomène connu sous le nom de *thrashing*. Les restarts, les heuristiques adaptatives et les algorithmes à cohérence forte sont des armes classiques de la PPC (Programmation Par Contraintes) développées pour lutter contre le *thrashing*.

1.1 Etat de l'art

Parmi les heuristiques de recherche, *min dom*, qui sélectionne les variables par ordre croissant de taille de domaine, a longtemps été considérée comme celle étant l'heuristique de backtracking la plus robuste. Récemment, des heuristiques *adaptatives* ont été introduites. Les deux premières heuristiques adaptatives génériques qui ont été introduites sont *impact* et *wdeg*. La première repose sur l'effet des décisions sur les domaines des variables, la deuxième associe un compteur à chaque contrainte (et indirectement à chaque variable) indiquant le nombre de fois où la contrainte a fait échouer la recherche. Des heuristiques à base de comptage et d'activité sont deux autres méthodes adaptatives récentes pour guider le processus de recherche.

2 Conflict ordering search

2.1 Last conflict

Le mécanisme de dernier conflit (LC, *last conflict*) peut être appliqué comme processus de réparation d'une heuristique de recherche à base d'ordre sur les variables. Plus précisément, la forme généralisée LC(k) [4] enregistre les k variables impliquées dans les dernières décisions qui ont provoqué un échec dans l'arbre de recherche après propagation, et les assigne en priorité. L'heuristique sous-jacente reprend la main une fois les k variables assignées.

2.2 Ordonner par dernier conflit

Nous introduisons un schéma générique pour guider les recherches arborescentes, la recherche par ordre de conflits (COS, Conflict Ordering Search). Celle-ci ordonne simplement les variables par ordre de dernier conflit, et choisit donc la variable non assignée qui a engendré un conflit le plus récemment. Si toutes les variables qui ont eu un conflit sont déjà assignées, la main est donnée à l'heuristique sous-jacente.

COS n'agit donc pas seulement comme un mécanisme de réparation de cette heuristique, mais finit par prendre complètement la main dessus, dès lors que toutes les variables ont provoqué un conflit.

2.3 Ensembles minimaux de variables en conflit

En PPC, lorsqu'un problème est infaisable, certains ensembles de variables ne peuvent pas être assignés sans que la propagation ne détecte un conflit. Un tel

ensemble de variables est dit *en conflit* ; cette notion est dépendante du filtrage.

Sur un problème infaisable, COS tente de dégager un sous-ensemble minimal (pour l'inclusion) de variables en conflit. En effet, une variable qui n'est pas en conflit finira par ne plus être sélectionnée par l'heuristique, puisqu'elle sera trop loin dans l'ordre de conflit.

Ce comportement est récursif : COS fait remonter un ensemble minimal de variables en conflit à la racine des sous-arbres infaisables, ce qui est utile aussi sur des problèmes faisables.

2.4 Combinaison avec les redémarrages

Les restarts se combinent bien avec COS : le fait que le dernier sous-ensemble en conflit minimal soit dans l'état interne gardé par COS lui permet de guider la recherche autour de ce conflit, ce qui tend à lui faire attaquer la partie difficile des problèmes proactivement.

3 Résultats

3.1 Recherche en arbre pure

Nous avons testé COS sur des instances à 120 tâches du RCPSP de la PSPLIB [2]. Nous avons utilisé de simples précédences binaires et un Time-Tabling pour les contraintes de ressource. COS est comparée aux heuristiques min/min, SetTimes et LC(k) pour le meilleur k trouvé expérimentalement. Une variante de COS qui mémorise les valeurs des décisions, COSPhase, a également été testée.

Intéressons-nous à la figure 1. Après une minute, il apparait clairement que COS et COSPhase dominent, mais la tendance (pente des courbes) est encore plus intéressante : min/min permet peu d'amélioration entre une seconde et une minute, alors que COS résoud environ 20 instances entre ces tops d'horloge.

3.2 Recherche avec redémarrages

Nous avons aussi testé COS dans un cadre de restarts/nogood [3], en forçant la recherche à redémarrer selon un nombre d'échecs croissant exponentiellement après avoir enregistré les nogoods dérivés de la recherche.

$\min^{rnd}$ / min est le pendant randomisé de min / min : elle choisit une variable parmi celles qui atteignent le minimum des non assignées. Les variantes "rst-" de COS et COSPhase, remettent l'état interne à zéro après chaque redémarrage. Elles permettent de voir l'effet du guidage proactif par les ensembles de conflit en haut de l'arbre. Les tests montrent un net avantage à garder l'état interne entre les redémarrages, comme indiqué par la figure 2.

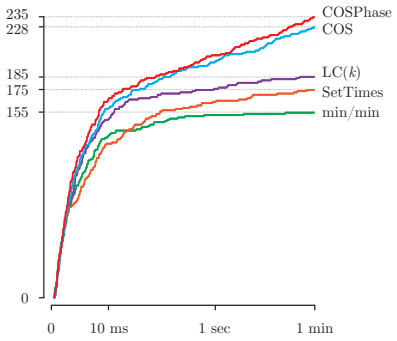


FIGURE 1 – Nombre d'instances résolues à l'optimal sous limite de temps. L'échelle des abscisses est logarithmique.

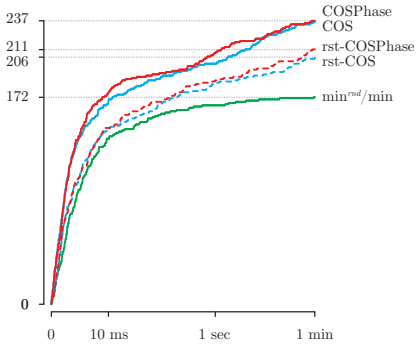


FIGURE 2 – Nombre d'instances résolues à l'optimal sous limite de temps. L'échelle des abscisses est logarithmique.

Références

[1] S. Gay, R. Hartert, C. Lecoutre, and P. Schaus. Conflict ordering search for scheduling problems. In *Proceedings of CP'15*. Springer International Publishing, 2015.

[2] R. Kolisch, C. Schwindt, and A. Sprecher. Benchmark instances for project scheduling problems. In *Project Scheduling*, pages 197–212. Springer, 1999.

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

[4] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Reasonning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173(18) :1592–1614, 2009.

Améliorations d'une approche de programmation par contraintes pour l'estimation de paramètres

Bertrand Neveu[1] **Martin de la Gorce**[1] **Gilles Trombettoni**[2]

[1] LIGM, Ecole des Ponts, UPEM, ESIEE Paris, CNRS, UPE, Champs-sur-Marne, France
[2] LIRMM, Université de Montpellier, France
Bertrand.Neveu@enpc.fr martin.delagorce@gmail.com Gilles.Trombettoni@lirmm.fr

Résumé

Cet article présente diverses améliorations d'une méthode combinatoire complète proposée par Jaulin, Walter et Didrit [2] pour trouver, dans le contexte de l'estimation de paramètres, toutes les instances d'un modèle paramétré compatibles avec au moins q observations dans les limites d'une tolérance donnée. Cette méthode a été appliquée à la détection de formes (cercles en 2D, plans en 3D). Cet article est un résumé de l'article publié dans les actes d'ICTAI 2015 [3].

1 Introduction

L'estimation de paramètres est un problème difficile qui apparaît souvent dans les sciences de l'ingénieur. Cela consiste à déterminer les n paramètres numériques d'un modèle à partir de m observations. L'étalonnage et la géolocalisation en sont des exemples. Un modèle paramétré est défini par une équation implicite $f(\mathbf{x}, \mathbf{p}) = 0$, $\mathbf{p}$ étant le vecteur de paramètres que l'on cherche à déterminer. Dans cet article, nous cherchons toutes les instances de modèle paramétré compatibles, dans les limites d'une tolérance donnée τ, avec au moins q observations parmi les m (plus exactement, une instance par ensemble maximal d'observations compatibles de cardinal supérieur ou égal à q). Nous appliquons cette méthode à la détection de formes géométriques comme des cercles en 2D et des plans en 3D, dans un nuage de points.

2 Etat de l'art

La principale méthode utilisée en vision par ordinateur et traitement d'image pour l'estimation de paramètres en présence d'observations aberrantes (*outliers*) est RANSAC (RANdom SAmple Consensus), méthode stochastique, procédant par tirages aléatoires successifs dans les observations pour déterminer les paramètres du modèle (n observations pour n paramètres), en cherchant à maximiser le nombre d'observations compatibles. Une version présentée dans [4] est adaptée à la détection de *plusieurs* objets, mais sans garantie de trouver toutes les solutions. Une première version d'un algorithme déterministe complet effectuant une recherche arborescente dans l'espace des paramètres a été proposée dans [2]. Le problème peut être formulé comme la recherche de solutions d'un CSP numérique, comprenant n variables réelles $p_1, \ldots, p_n$ ayant chacune pour domaine un intervalle et liées par la contrainte globale indiquant qu'au moins q observations sont compatibles, l'ensemble C représentant les observations o_i compatibles.

$$C(p_1, \ldots, p_n) = \{\mathbf{o}_i | abs(f(\mathbf{o}_i, p_1, \ldots, p_n)) \leq \tau\} \quad (1)$$

$$|C((p_1, \ldots, p_n)| \geq q \quad (2)$$

Chaque nœud de l'arbre de recherche correspond à une partie de l'espace exploré, c.-à-d. une boîte $[p_1] \cdots [p_n]$. On utilise un opérateur de contraction (filtrage) basé sur la q-intersection des observations pour réduire les intervalles sur chacune des dimensions de la boîte courante. Dans l'espace des paramètres, pour chaque observation, l'ensemble des points compatibles dans la boîte courante est tout d'abord approximé par une boîte. La q-intersection de ces boîtes est la boîte enveloppe de l'union de toutes les intersections de q de ces boîtes. Trouver la q-intersection exacte est un problème DP-complet [1], aussi nous avons utilisé l'algorithme non optimal classique de projection sur les différentes dimensions. Cet algorithme, de complexité totale en $O(nm \log(m))$, permet de détecter

des intervalles ne pouvant pas être compatibles avec q observations et donc de réduire l'intervalle de la boîte courante dans cette dimension. Nous sommes partis de la méthode de [2] implantée dans Ibex et l'avons améliorée sur les points suivants.

3 Améliorations de la méthode

Nous avons apporté à l'algorithme complet de recherche de solutions des améliorations génériques (indépendantes de l'application de détection de plans ou de cercles) et des améliorations spécifiques pour ces applications.

3.1 Améliorations génériques

Validation des solutions. Dans l'arbre de recherche, nous maintenons l'ensemble des observations possibles sur la boîte courante et des observations valides en un point de cette boîte (sous ensemble des possibles). Une condition d'arrêt de l'algorithme dans la branche courante est atteinte quand ces ensembles sont les mêmes.

L'ensemble des observations possibles est mis à jour directement par l'algorithme de q-intersection.

q-intersection supplémentaire. L'algorithme de contraction par q-intersection procède par projections des observations sur les différentes dimensions de la boîte courante. Nous réalisons une projection sur une direction supplémentaire dans laquelle nous espérons obtenir des petits intervalles et aboutir à une q-intersection vide. Pour cela, nous linéarisons les équations de chaque observation en les relaxant et projetons les parallélogrammes obtenus sur la direction correspondant à la moyenne de leurs normales (Figure 1).

3.2 Améliorations spécifiques

Contraintes dédiées. Au lieu d'utiliser l'algorithme général de contraction effectuant des calculs sur la représentation des fonctions mathématiques en Ibex, nous avons réécrit ces calculs pour les contraintes simples (linéaires ou quadratiques) utilisées dans notre modélisation.

Heuristique de bissection. On bissecte en premier les domaines des variables de direction pour les plans et de coordonnées du centre pour les cercles, et on termine par la distance à l'origine pour les plans et le rayon pour les cercles.

Modélisation linéaire. Pour la détection de plans, nous proposons une modélisation linéaire des contraintes liant les paramètres, ce qui permet des calculs plus efficaces.

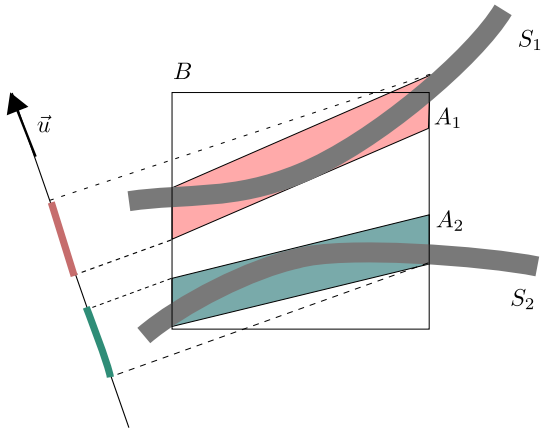


Figure 1: Projection des parallélogrammes A_i, relaxations linéaires des zones de compatibilité S_i, sur la direction de la moyenne de leurs normales.

4 Validation expérimentale

Nous avons testé notre algorithme sur des problèmes de détection de plans sur des exemples artificiels et sur une scène réelle ainsi que sur la détection de cercle en 2D pour trouver une bouée dans une image sous-marine. Les expérimentations ont montré que chacune des améliorations apportait un gain en temps de calcul significatif. Sur chaque instance, 2 à 3 ordres de grandeur ont été gagnés en tout par rapport à l'algorithme originel.

References

[1] C. Carbonnel, G. Trombettoni, P. Vismara, and G. Chabert. Q-intersection algorithms for constrained-based robust parameter estimation. In *Proc. of AAAI 2014*, pages 2630–2636, 2014.

[2] L. Jaulin, E. Walter, and O. Didrit. Guaranteed robust nonlinear parameter bounding. In *CESA'96 IMACS Multiconference (Symposium on Modelling, Analysis and Simulation)*, pages 1156–1161, 1996.

[3] B. Neveu, M. de la Gorce, and G. Trombettoni. Improving a constraint programming approach for parameter estimation. In *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*, pages 852–859, 2015.

[4] R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, 26(2):214–226, 2007.

Revisiter les contraintes de stabilité bi-latérale *

Mohamed Siala **Barry O'Sullivan**

Insight Centre for Data Analytics
Department of Computer Science, University College Cork, Ireland
{mohamed.siala,barry.osullivan}@insight-centre.org

Résumé

Ce papier résume les contributions présentées dans [5]. Nous montrons d'abord que les approches existantes de la littérature pour les contraintes de stabilité bi-latérale n'assurent pas l'arc consistance, mais plutôt la $\mathcal{D}$-consistance de bornes (BC(D)). Nous proposons ensuite un algorithme pour la $\mathcal{D}$-consistance de bornes avec une complexité temporelle linéaire. Nous proposons également une adaptation du filtrage pour maintenir l'arc consistance. En plus, nous introduisons le premier algorithme polynomial pour résoudre le problème d'affectation de résidents aux hôpitaux en présence de paires interdites et forcées. Finalement, nous montrons que la complexité temporelle pour résoudre le dernier problème pour le cas de mariage stable peut être réduite à $O(n^2)$.

1 Introduction & notations

Dans les problèmes de stabilité bi-latérale, on cherche à trouver un couplage entre deux ensembles d'agents tout en respectant un critère de stabilité. Exprimer ce genre de problèmes en PPC est extrêmement bénéfique pour résoudre différentes variantes.

Dans le problème d'affectation de résidents aux hôpitaux (appelé admission à l'université dans [2]), le but est d'associer des résidents $r_1, \ldots, r_{n_R}$ à un ensemble d'hôpitaux $h_1, \ldots, h_{n_H}$. Chaque hôpital h_j a une capacité c_j définissant le nombre maximum de résidents affectés. Chaque résident r_i est associé a une liste d'entiers $\mathcal{R}_i$ pour classer un sous ensemble d'hôpitaux par ordre strict de préférence. D'une façon similaire, chaque hôpital h_j est associé à une liste de préférence $\mathcal{H}_j$ pour classer les résidents.

Soit $\mathcal{E} = \{(i,j) \mid i \in [1,n_R] \wedge j \in [1,n_H] \wedge i \in \mathcal{H}_j \wedge j \in \mathcal{R}_i\}$ l'ensemble des paires acceptables. Un *couplage* $\mathcal{M}$ est un sous ensemble de $\mathcal{E}$ tel que $|\{j \mid (i,j) \in \mathcal{M}\}| \leq 1$ $\forall i \in [1,n_R]$ et $|\{i \mid (i,j) \in \mathcal{M}\}| \leq c_j, \forall j \in [1,n_H]$. Une paire $(i^*,j^*) \in \mathcal{E} \setminus \mathcal{M}$ bloque $\mathcal{M}$ ssi :

1. $|\{j \mid (i^*,j) \in \mathcal{M}\}| = 0$, ou $\exists k \in [1,n_H]$ tel que $(i^*,k) \in \mathcal{M}$ et r_{i^*} préfère h_{j^*} à h_k.
2. $|\{i \mid (i,j^*) \in \mathcal{M}\}| < c_{j^*}$, ou $\exists l \in [1,n_R]$ tel que $(l,j^*) \in \mathcal{M}$ et h_{j^*} préfère r_{i^*} à r_l.

Un couplage $\mathcal{M}$ est stable si et seulement s'il n'existe aucune paire qui bloque $\mathcal{M}$. Le problème d'affectation de résidents aux hôpitaux (*HR*) consiste à trouver un couplage stable. Le mariage stable est un cas particulier de ce problème tel que $c_j = 1$ pour tout $j \in [1,n_H]$. Le problème HR peut être résolu avec une complexité temporelle de $O(L)$ grâce à Gale et Shapley [2].

On dit qu'une contrainte est borne-$\mathcal{D}$-consistante (*BC(D)*) si pour chaque variable x_i qu'elle met en jeu, la borne inférieure (resp. supérieure) de x_i appartient à une affectation valide pour cette contrainte. Il est important de noter que cette propriété est plus forte que la consistance de bornes classique.

2 Résumé des contributions

Nous résumons les contributions présentées dans [5].

2.1 Caractérisation du niveau de filtrage

Notons d'abord que toutes les propositions de la littérature (par exemple [3, 4]) sont équivalentes en terme de filtrage. Soit Γ le premier modèle PPC introduit dans [4]. Dans Γ, chaque résident r_i est associé à une variable x_i dont le domaine représente le rang de chaque hôpital dont l'index appartient à $\mathcal{R}_i$. Nous avons remarqué que ce modèle n'est pas complet en terme de propagation (voir l'exemple 1 dans [5]).

Nous avons introduit la contrainte globale 2-SidedStability$(\mathcal{X},\mathcal{A},\mathcal{B},\mathcal{C})$ de la façon suivante :

*Travail soutenu par Science Foundation Ireland (SFI), référence SFI/12/RC/2289.

— $\mathcal{X}$ est l'ensemble des variables $x_1, \ldots, x_{n_R}$ définies de la même façon que dans Γ
— $\mathcal{A} = \{\mathcal{R}_1, \ldots, \mathcal{R}_{n_R}\}$
— $\mathcal{B} = \{\mathcal{H}_1, \ldots, \mathcal{H}_{n_H}\}$
— $\mathcal{C} = \{c_1, \ldots, c_{n_H}\}$
— 2-SidedStability est satisfiable ssi le couplage correspondant est stable.

Nous avons montré que si Γ est *AC* alors la $\mathcal{D}$-consistance de bornes est maintenue. Les lemmes suivants sont utilisés pour montrer ce résultat.

Lemme 2.1 *Si Γ est AC alors assigner chaque variable à sa valeur minimale constitue une solution.*

Lemme 2.2 *Si Γ est AC alors assigner chaque variable à sa valeur maximale constitue une solution.*

2.2 Un propagateur BC(D) avec une complexité linéaire

Dans cette partie nous avons introduit un nouveau propagateur BC(D) pour la contrainte 2-SidedStability avec une complexité temporelle inférieure à la meilleur borne supérieure connue. Notre propagateur est basé sur le théorème suivant.

Théorème 2.1 2-SidedStability$(\mathcal{X}, \mathcal{A}, \mathcal{B}, \mathcal{C})$ *est BC(D) ssi assigner chaque variable à son maximum (respectivement minimum) constitue une solution.*

Nous avons montré que la propriété du Théorème 2.1 peut être maintenue en $O(L)$ tout au long d'une branche de l'arbre de recherche, ce qui améliore la meilleure complexité connue [4] par un facteur de $c = \max\{c_j \mid j \in [1, n_H]\}$.

2.3 L'arc consistance

Nous avons présenté une méthode naive pour l'arc consistance. D'abord la consistance BC(D) est maintenue. Ensuite, pour chaque variable, nous effectuons la BC(D) sur le nouveau domaine où la borne supérieure est supprimée. Nous itérons le processus jusqu'à ce qu'on atteint la borne inférieure. Une valeur appartient à une affectation valide ssi elle n'est pas filtrée pendant ce processus.

Notons que l'algorithme n'est pas incrémental et que chaque appel a un coût de $O(n_R \times L)$.

2.4 Le cas avec des paires interdites et forcées

Dans la variante du problème d'affectation de résidents aux hôpitaux en présence de paires interdites et forcées (*HRFF*), on cherche à trouver un couplage stable qui inclut et/ou exclut un certain nombre de paires. Il n'existe aucun algorithme pour résoudre ce problème dans la littérature. Cette observation reste vraie même s'il n'y a aucune paire forcée [4].

En utilisant le Théorème 2.1, nous avons montré que le problème est en effet polynomial et peut être résolu en $O(L)$.

Le meilleur algorithme pour résoudre le cas particulier de ce probleme pour le mariage stable a une complexité temporelle de $O(n^4)$ [1], où n est le nombre d'hommes/femmes. En utilisant notre approche, il est évident que ce problème peut être résolu en $O(n^2)$.

3 Conclusion

Nous avons étudié l'aspect de filtrage pour la contrainte 2-SidedStability. Tout d'abord, nous avons montré que le niveau de filtrage pour toutes les approches existantes de la littérature est BC(D). Ensuite, nous avons proposé un algorithme optimal pour la $\mathcal{D}$-consistance de bornes ainsi qu'une adaptation de ce dernier pour maintenir l'arc consistance. Nous avons aussi montré, pour la première fois, que la variante du problème HR en présence de paires interdites et forcées est polynomiale. Finalement, nous avons amélioré la complexité temporelle de ce dernier problème pour le cas du mariage stable par un facteur de n^2.

Notre étude expérimentale a montré essentiellement que les performances de l'algorithme BC(D) sont nettement meilleures que celle d'AC sur une variante du problème avec couples. Ce comportement est dû au fait que notre algorithme d'AC n'est pas incrémental.

Références

[1] Vânia M. F. Dias, Guilherme Dias da Fonseca, Celina M. Herrera de Figueiredo, and Jayme Luiz Szwarcfiter. The stable marriage problem with restricted pairs. *Theor. Comput. Sci.*, 306(1-3) :391–405, 2003.

[2] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *American mathematical monthly*, pages 9–15, 1962.

[3] Ian P. Gent, Robert W. Irving, David Manlove, Patrick Prosser, and Barbara M. Smith. A constraint programming approach to the stable marriage problem. In *Proceedings of CP*, pages 225–239, 2001.

[4] David Manlove, Gregg O'Malley, Patrick Prosser, and Chris Unsworth. A constraint programming approach to the hospitals / residents problem. In *Proceedings of CPAIOR*, pages 155–170, 2007.

[5] Mohamed Siala and Barry O'Sullivan. Revisiting two-sided stability constraints. In *Proceedings of CPAIOR*, 2016.

Index des auteurs

Index des mots clefs