



Actes des 20es Journées Francophones de Programmation par Contraintes

Hélène Verhaeghe

► To cite this version:

Hélène Verhaeghe. Actes des 20es Journées Francophones de Programmation par Contraintes. Plate-Forme Intelligence Artificielle, Jul 2025, Dijon, France. Association Française pour l'Intelligence Artificielle, 2025. hal-05189725v1

HAL Id: hal-05189725

<https://hal.science/hal-05189725v1>

Submitted on 28 Jul 2025 (v1), last revised 31 Jul 2025 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons CC BY-NC-ND 4.0 - Attribution - Non-commercial use - No
Derivative Works - International License



AfIA

Association française
pour l'Intelligence Artificielle

JFPC

*20^{es} Journées Francophones
de Programmation par Contraintes*

PFIA 2025



Table des matières

Hélène Verhaeghe

Éditorial	5
Comité de programme	6
Session 1, lundi 30 juin, 10h45-12h00 : Applications	7
E. Legrand, V. Coppé, D. Catanzaro, P. Schaus	
Une approche de programmation dynamique pour le problème de séquençement des tâches et de changement d'outil (article résumé)	8
L. Ligny, G. Picard, S. Roussel, E. Vareilles	
Configuration de flottes de robots hétérogènes pour la planification de mission : une approche basée programmation par contraintes	10
T. Giachetto de Araujo, Z. Zheng, M. Py, S. Cherif, R. Sá Shibasaki, L. Deroussi, N. Grangeon, C.-M. Li	
Approche heuristique pour l'équilibrage de lignes d'assemblage avec minimisation du pic de consommation énergétique	20
O. Bailleux	
Quel codage CNF pour les contraintes de concordance de motifs d'Eternity II ?	29
Session 2, lundi 30 juin, 15h00-16h00 : Applications	37
A. Burlats, P. Schaus, C. Pelsser	
Sélection quasi-optimale de routes pour la surveillance de réseaux à l'aide de la génération de colonnes	38
D. Anh Le, S. Roussel, C. Lecoutre	
Équilibrage de ligne d'assemblage aéronautique sous contrainte de ressources avec effet d'apprentissage : une approche de programmation par contraintes	43
D. Sprockeels, P. Van Roy	
Le Projet Harmoniser : un Outil Pratique à Quatre Couches pour la Composition Musicale Basé sur les Contraintes	49
Session 3a, lundi 30 juin, 16h30-17h20 : Heuristiques	54
A. Delecluse, P. Schaus	
Heuristiques boîte-noire de sélection de valeur pour les problèmes d'optimisation en programmation par contraintes (article résumé)	55
Q. Cappart, L. Boisvert, T. Marty, T. François, P. Tessier, L. Gautier, L.-M. Rousseau	
Apprentissage et calibrage d'une heuristique générique de choix de valeur dans un solveur de programmation par contraintes (article résumé)	57
C. Lecoutre	
Search is not Dead !	60
Session 3b, lundi 30 juin, 17h20-18h00 : Applications	64
A. Khoualdia, S. Cherif, S. Devismes, L. Robert	
Analyse de l'autostabilisation de l'unisson synchrone via la satisfiabilité propositionnelle	65
C. Berton, S. Cherif, C. Delaplace	
Satisfiabilité pour le décodage par syndrome	75
Session 4, mardi 1 juillet, 10h30-11h00 : PPC & Apprentissage Automatique	83
H. Verhaeghe, Q. Cappart, G. Pesant, C.-G. Quimper	

Apprentissage de précédences pour des problèmes de planification avec des réseaux de neurones en graphe (article résumé)	84
S. Bessa, D. Dabert, M. Bourgeat, L.-M. Rousseau, Q. Cappart	
Apprentissage de bornes duales valides en programmation par contrainte : Décomposition lagrangienne amplifiée avec apprentissage auto-supervisé (article résumé)	87
 Session 5, mardi 1 juillet, 15h00-16h00 : MaxSAT	90
Z. Zheng, S. Cherif, R. Sá Shibasaki, C.-M. Li	
Approches exactes pour le problème de satisfiabilité diversifiée	91
J. Zhang, C.-M. Li, S. Cherif, S. Li, Z. Zheng	
Prétraitement PLNE pour la simplification des instances MaxSAT	101
Z. Zheng, S. Cherif, R. Sá Shibasaki, C.-M. Li, J. Zhang	
Formulations MaxSAT pour la programmation non linéaire en nombres entiers.....	106
 Session 6, mardi 1 juillet, 16h30-17h30 : PPC & Apprentissage Automatique	117
A. Beauchamp, T.-B.-H. Dao, S. Loudni, C. Vrain	
Ensemble de correction minimal basé sur l'adéquation des contraintes en clustering	118
H. Barral, Q. Cappart, M.-J. Huguet, S. Thiébaux	
Convergence accélérée de la Décomposition Lagrangienne pour la Planification Automatique	128
R.-N. Assaf, F. Lardeux, F. Saubion	
Graphes de positions : représentations d'alignements et recherche de motifs	134
 Session 7a, mercredi 2 juillet, 10h30-11h25 : Contraintes	144
M. Schmied, J.-C. Régis	
Amélioration de la contrainte globale de cardinalité avec coûts par l'introduction de points de repère (article résumé).....	145
V. Allard, V. Barichard, E. Monfroy, J.-M. Richer	
Activation de règles CHR pour la composition et l'orchestration de solveurs par contraintes.	147
P. Talbot	
Réseau de Contrainte Ternaire pour une Propagation Efficace de Bornes sur GPU.....	151
 Session 7b, mercredi 2 juillet, 11h25-12h00 : Tests & Vérifications.....	163
W. Vanroose, I. Bleukx, J. Devriendt, D. Tsouros, H. Verhaeghe, T. Guns	
Tests à données aléatoires par mutations pour les systèmes de programmation par contraintes (article résumé).....	164
C. Dubois	
Domaines formellement vérifiés	167
 Session 8, mercredi 2 juillet, 15h30-16h20 : Applications.....	171
H. Willot, J.-L. Farges, G. Picard, P. Pavero	
Gestion des ressources de communication pour satellites dans une fédération de constellations d'observation de la Terre (article résumé)	172
X. Pucel, S. Roussel	
Modèle de programmation par contraintes pour l'équilibrage et l'ordonnancement des lignes d'assemblage avec travailleurs mobiles et stations parallèles (article résumé)	174
D. Olivier Fernandez Pons, P. Vilim	
Remplissage de camions citerne, routage et gestion d'inventaire d'hydrocarbures avec OptalCP	176

Éditorial

20^{es} Journées Francophones de Programmation par Contraintes

Les Journées Francophones de Programmation par Contraintes (JFPC) sont organisées à l'initiative de l'Association Française de Programmation par Contraintes (AFPC). Elles constituent le principal congrès francophone centré autour des problèmes de satisfaction de contraintes (CSP), de satisfiabilité (SAT) et de programmation logique sous contraintes (CLP). Les JFPC regroupent aussi des thématiques liées comme la recherche opérationnelle (RO), les méta-heuristiques, l'analyse par intervalles, etc. De part ses applications, la programmation par contraintes s'ouvre à de nombreuses communautés connexes, en particulier la robotique et la bio-informatique.

Les JFPC se veulent un lieu convivial de rencontres, de discussions et d'échanges entre doctorants, chercheurs confirmés et industriels. Historiquement, les JFPC sont issues de la fusion entre les Journées Francophones de Programmation Logique avec Contraintes (JFPLC) qui existaient depuis 1992 (voire 1982, si on considère son prédécesseur, le Séminaire Programmation en Logique) et des Journées Nationales sur la Résolution Pratique de Problèmes NP-Complets (JNPC) qui existaient depuis 1995 (pendant les deux premières années sous le nom de CNPC). Ces vingtièmes Journées Francophones de Programmation par Contraintes ont eu lieu du 30 juin au 2 juillet 2025 à Dijon pour la troisième fois dans le cadre de la Plate-Forme Intelligence Artificielle (PFIA'25). Auparavant, les JFPC avaient pendant quelques années été organisées conjointement avec les JFPLC (avant la fusion) et plus tard avec les JIAF (avant que ces derniers ne rejoignent la PFIA).

Le comité de programme a été composé en essayant de représenter le plus de lieux possibles et en intégrant un certain nombre de jeunes chercheurs (doctorants en toute fin de thèse ou ATER) en plus des chercheurs expérimentés. La présence de jeunes chercheurs dans le comité de programme se justifie par l'état d'esprit général de JFPC : être un lieu de rencontres entre tous les chercheurs de la communauté où les plus jeunes peuvent prendre de l'expérience dans un cadre convivial et bienveillant. Le passé a montré que les chercheurs débutants sont très soucieux de relire les articles avec un soin particulier, ce qui compense leur manque d'expérience. Je remercie tous les membres du comité de programme du temps qu'ils ont passé et de la qualité de leur relecture.

Cette année, trente soumissions (douze articles longs, neuf articles courts et neuf résumés d'articles déjà publiés dans une conférence internationale de renom l'an passé) ont été relues, presque toutes ont été acceptées (une a été rejetée, jugée hors thème et une a malheureusement été retirée, ses auteurs ne pouvant finalement pas participer à PFIA). Ce taux inédit (et difficilement dépassable) d'acceptation ne résulte pas seulement du caractère traditionnellement peu sélectif de la conférence (qui a par exemple tendance à accepter le travail non complètement abouti d'un doctorant afin de lui permettre de présenter son travail devant la communauté et d'avoir des retours constructifs). Il s'avère qu'il n'y a eu réellement aucune note globale négative attribuée à aucun papier.

En plus des présentations orales des articles que vous retrouverez dans ces actes, nous avons eu pendant la conférence deux exposés invités : celui de Louis-Martin Rousseau intitulé "En route vers des solveurs neuro-symbolique en programmation par contrainte" (invité PFIA) et celui de Sylvie Thiebaut intitulé "Graph Learning for Planning" (invitée commune avec la journée GdR RADIA).

Il me reste à remercier particulièrement toute l'équipe du comité d'organisation de PFIA qui a su répondre aux besoins de JFPC au-delà de mes attentes et qui nous ont accueilli avec bienveillance et souplesse, l'encadrement rigoureux et très rodé de l'organisation ne prenant jamais le dessus sur leur souci d'essayer de satisfaire au maximum nos demandes et nos particularités.

Hélène Verhaeghe

Comité de programme

Présidence

- Hélène Verhaeghe, UCLouvain, Belgique.

Membres

- Quentin Cappart, Polytechnique Montréal, Canada
- Clément Carbonnel, LIRMM-CNRS Montpellier, France
- Sami Cherif, Université de Picardie Jules Verne, France
- Thi-Bich-Hanh Dao, University of Orleans, France
- Augustin Delecluse, UCLouvain, Belgique
- Guillaume Derval, ULiège, Belgique
- Gaël Glorian, Huawei Technologies, France
- Jin-Kao Hao, Université of Angers, France
- Vinasétan Ratheil Houndji, Université d'Abomey-Calavi, Bénin
- Marie-José Huguet, LAAS-CNRS Université de Toulouse, France
- Roger Kameugne, University of Maroua, Cameroun
- George Katsirelos, INRAE Paris, France
- Jean-Marie Lagniez, CRIL-CNRS Université d'Artois, France
- Arnaud Lallouet, Huawei Technologies, France
- Olivier Lhomme, IBM, France
- Chu-Min Li, Université de Picardie Jules Verne, France
- Margaux Nattaf, Grenoble INP, France
- Bertrand Neveu, Ecole des ponts ParisTech, France
- Abdelkader Ouali, University of Caen Normandy, France
- Xiao Peng, INSA Lyon, France
- Eric Piette, UCLouvain, Belgique
- Cédric Pralet, ONERA Toulouse, France
- Nicolas Prcovic, LIS-CNRS Université Aix Marseille, France
- Charles Prud'Homme, IMT Atlantique, France
- Mathieu Py, LIMOS-CNRS University Clermont Auvergne, France
- Stéphanie Roussel, ONERA, France
- Suruthy Sekar, Huawei Technologies, France
- Elise Vareilles, University Mines Albi, France
- Thierry Vidal, Technical University of Tarbes, France
- Julien Vion, LAMIH-CNRS Université Polytechnique Hauts de France, France
- Romain Wallon, CRIL-CNRS Université d'Artois, France

Session 1, lundi 30 juin, 10h45-12h00 : Applications

Une approche de programmation dynamique pour le problème de séquençement des tâches et de changement d'outil

Emma Legrand¹, Vianney Coppé², Daniele Catanzaro³, Pierre Schaus¹

¹ ICTEAM, UCLouvain, Belgique

² Hexaly, France

³ CORE, UCLouvain, Belgique

25 février 2025

Résumé

Nous présentons un algorithme exact basé sur la programmation dynamique pour le problème de séquençement des tâches et de changement d'outil, un problème combinatoire issu des systèmes de fabrication. Nous introduisons une nouvelle famille de bornes inférieures plus serrées que celles existantes, améliorant la qualité des solutions et l'efficacité de l'élagage. Nous utilisons A* et ses variantes anytime pour explorer l'espace des solutions, ainsi qu'une structure de données appelée FreeTools pour suivre l'état et calculer efficacement les coûts incrémentaux. Nos expériences montrent que notre approche surpasse les méthodes de l'état de l'art, notamment la programmation linéaire en nombres entiers.

Mots-clés

Optimisation combinatoire, séquençement des tâches, changement d'outil, séparation et évaluation, A*, programmation dynamique.

Abstract

We present an exact algorithm based on dynamic programming for the Job Sequencing and Tool Switching Problem (JS-TSP), a combinatorial problem derived from manufacturing systems. We introduce a new family of lower bounds that are tighter than existing ones, improving solution quality and pruning efficiency. We use A* and its anytime variants to explore the solution space and a data structure called FreeTools to track state and compute incremental costs efficiently. Our experiments show that our approach outperforms state-of-the-art methods, notably integer linear programming.

Keywords

Combinatorial optimization, job sequencing, tool switching, branch and bound, A*, dynamic programming.

1 Introduction

Le problème de séquençement des tâches et de changement d'outil est un problème d'optimisation combinatoire issu des systèmes de fabrication. Il consiste à ordonnancer une séquence de tâches sur une machine flexible $\mathcal{M}$ qui

peut être configurée avec un ensemble de maximum c outils parmi un ensemble $T = 0, 1, 2, \dots, m-1$ de $m \geq 3$ outils. Nous appelons c la *capacité* de $\mathcal{M}$. De plus, nous considérons un ensemble $J = 1, 2, \dots, n$ de $n \geq 3$ tâches à exécuter séquentiellement sur $\mathcal{M}$. Chaque tâche $j \in J$ nécessite un ensemble d'outils $t(j) \subseteq T$, avec $|t(j)| \leq c$. Lorsque $\mathcal{M}$ passe d'une tâche à une autre, il est nécessaire de choisir quels outils conserver et lesquels remplacer, cette opération est appelée *changement d'outil*.

Ce problème consiste à trouver une séquence de tâches qui minimise le nombre total de changements d'outils nécessaires. Tang et Denardo ont démontré en 1987 qu'il est *NP-difficile* et ont proposé la première formulation en programmation linéaire en nombres entiers (ILP) [7]. D'autres approches ont été développées [4, 2, 1], notamment des méthodes *Branch-and-Bound* (B&B) [5]. Nous nous penchons uniquement sur les méthodes de résolution exactes. Un récent article [6] a été publié sur ce sujet. Cependant, il n'a pas été inclus dans la comparaison car il a été publié après la soumission de l'article.

Tâches	1	2	3	4	5	6
Outils	0	1	1	0	5	3
	1	3	5	4	6	5
	2	4	6	6	7	
			8	8		
Capacité : 4						

TABLE 1 – Exemple d'une instance avec 9 outils et 6 jobs.

Tâches	1	2	4	3	5	6
Outils	<u>0</u>	①	0	<u>1</u>	6	<u>3</u>
	<u>1</u>	1	<u>6</u>	6	<u>7</u>	5
	<u>2</u>	<u>3</u>	8	8	5	
		<u>4</u>	4	<u>5</u>		
Capacité : 4						

TABLE 2 – Exemple d'une solution optimale pour l'instance 1 pour laquelle la valeur optimale est égale à 11.

Par exemple, la Table 2 montre la séquence de travail qui minimise le nombre total de changements d'outils pour l'instance présentée dans la Table 1. La valeur optimale pour cette instance est égale à 11. Les outils soulignés sont ceux qui donnent lieu à un changement d'outil, tandis que les outils encadrés ont la particularité de ne pas être immédiatement nécessaires pour traiter un travail spécifique j dans la séquence, mais d'être nécessaires pour traiter les tâches qui suivent j . Cette propriété, introduite pour la première fois par [7], donne lieu à un algorithme gourmand, communément appelé *Keep Tool Needed Soonest* (KTNS), qui calcule en un temps polynomial les changements d'outils optimaux pour une séquence de travail fixe. Cependant, ces approches restent limitées pour des grandes instances.

2 Contributions et conclusion

Nos contributions principales incluent l'introduction d'une nouvelle famille de bornes inférieures, qui est plus serrée que les bornes existantes [3, 5]. Cette famille de bornes combine les avantages des bornes déjà présentes dans l'état de l'art en utilisant une approche basée sur les arbres couvrants minimaux (MST). Dans ces arbres, les noeuds représentent une tâche ou un ensemble de tâches et les arêtes représentent le coût minimum pour passer d'une tâche à une autre. Cette amélioration permet de réduire significativement l'espace de recherche en élaguant plus efficacement les noeuds non prometteurs. Nous proposons également une version incrémentale de l'algorithme KTNS, appelée Lazy-KTNS, qui calcule de manière incrémentale le coût des séquences partielles. Cet algorithme utilise une structure de données spécifique, appelée FreeTools, pour garder une trace des outils pouvant être réutilisés gratuitement dans les étapes ultérieures. Cette approche réduit la complexité temporelle de $\mathcal{O}(n^2 \cdot m)$ à $\mathcal{O}(n \cdot c^2)$.

La Figure 1 permet d'observer que pour la majorité des instances l'algorithme Lazy-KTNS prend moins de temps pour trouver et prouver l'optimalité que l'algorithme KTNS.

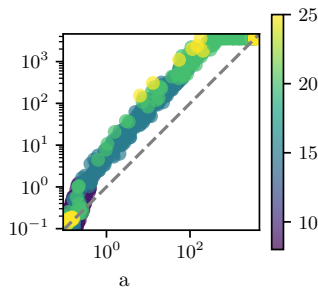


FIGURE 1 – Comparaison de la méthode B&B avec les algorithmes KTNS et Lazy-KTNS (en sec).

Enfin, nous reformulons le problème en termes de recherche en graphe, en utilisant l'algorithme A* et ses variantes anytime (comme Anytime Column Search et Iterative Beam Search). Cette reformulation permet d'éviter l'exploration redondante des séquences partielles équivalentes, ce qui réduit le nombre de nœuds explorés. Nous

montrons que cette approche est plus efficace que les méthodes de B&B traditionnelles, en particulier pour les instances de grande taille et avec Anytime Column Search.

Nos expériences montrent que notre approche surpasse les méthodes existantes, y compris les formulations ILP, en termes de temps de calcul et de capacité à résoudre des instances de grande taille. Par exemple, notre méthode est capable de résoudre 100% des instances jusqu'à 15 jobs en moins d'une heure, alors que les formulations ILP existantes n'y arrivent pas.

En conclusion, notre travail propose une nouvelle approche pour résoudre le problème de séquençage des tâches et de changement d'outil, en combinant des bornes inférieures améliorées, un algorithme de calcul incrémental des coûts et une reformulation en termes de recherche en graphe. Ces contributions permettent de résoudre efficacement des instances de grande taille, ce qui était auparavant difficile avec les méthodes existantes. Pour des travaux futurs, nous envisageons d'explorer des variantes de la borne et d'étudier l'application de techniques de programmation par contraintes pour résoudre ce problème.

Références

- [1] Christian Bessiere, Emmanuel Hebrard, Marc-André Ménard, Claude-Guy Quimper, and Toby Walsh. Buffered resource constraint : Algorithms and complexity. In *Integration of AI and OR Techniques in Constraint Programming : 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings 11*, pages 318–333. Springer, 2014.
- [2] Daniele Catanzaro, Luis Gouveia, and Martine Labbé. Improved integer linear programming formulations for the job sequencing and tool switching problem. *European journal of operational research*, 244(3) :766–777, 2015.
- [3] Yves Crama, Alwin G Oerlemans, Frits CR Spieksma, Yves Crama, Alwin G Oerlemans, and Frits CR Spieksma. *Minimizing the number of tool switches on a flexible machine*. Springer, 1996.
- [4] Gianpaolo Ghiani, Antonio Grieco, and Emanuela Guerriero. Solving the job sequencing and tool switching problem as a nonlinear least cost hamiltonian cycle problem. *Networks : An International Journal*, 55(4) :379–385, 2010.
- [5] Gilbert Laporte, Juan Jose Salazar-Gonzalez, and Frederic Semet. Exact algorithms for the job sequencing and tool switching problem. *IIE transactions*, 36(1) :37–45, 2004.
- [6] Yuzhuo Qiu, Mikhail Cherniavskii, Boris Goldengorin, and Panos M. Pardalos. A computational study of the tool replacement problem. *INFORMS Journal on Computing*.
- [7] C. S. Tang and E. V. Denardo. Models arising from a flexible manufacturing machine, Part I : Minimization of the number of tool switches. *Operations Research*, 36(5) :767–777, 1987.

Configuration de flottes de robots hétérogènes pour la planification de mission: une approche basée programmation par contraintes

Lucas Ligny^{1,2}, Gauthier Picard¹, Stéphanie Roussel¹, Elise Vareilles²

¹ DTIS, ONERA, Université de Toulouse, France

² Centre Génie Industriel, Université Toulouse, IMT Mines Albi, Albi, France

{lucas.ligny,gauthier.picard,stephanie.roussel}@onera.fr elise.vareilles@mines-albi.fr

Résumé

Avec l'autonomie croissante des robots et la variété des tâches qu'ils sont désormais en mesure de réaliser, ceux-ci sont de plus en plus souvent utilisés dans le cadre de missions complexes et critiques. Cet article s'intéresse au problème de la configuration des flottes de robots pour la planification de mission. Dans nos travaux, la planification d'une mission, avant sa phase d'exécution, repose sur la résolution de trois problèmes de configuration intrinsèquement liés : le problème de configuration d'agents, noté ACP, le problème de configuration de la flotte d'agents, noté FCP et le problème de configuration du plan de mission, noté PCP. Dans cet article, nous considérons la planification de mission comme un seul et unique problème, composé de trois sous-problèmes interdépendants. Un modèle de programmation par contraintes (PPC) est proposé pour le problème global. Notre modélisation est évaluée sur le solveur IBM ILOG CP Optimizer et sur soixante-quatre instances de type Recherche & Sauvetage générées artificiellement.

Mots-clés

Configuration, flotte hétérogène, PPC, Search & Rescue.

Abstract

With the increasing autonomy of robots and the variety of tasks they are now able to perform, they are increasingly used in complex and critical missions. This paper focuses on the problem of configuring robot fleets for mission planning. In our work, planning a mission, before its execution phase, is based on solving three intrinsically linked configuration problems : the agent configuration problem, denoted ACP, the agent fleet configuration problem, denoted FCP and the mission plan configuration problem, denoted PCP. In this paper, we consider mission planning as a single problem, composed of three interdependent subproblems. A constraint programming (CP) model is proposed for the global problem. Our model is evaluated on the IBM ILOG CP Optimizer solver and on sixty-four artificially generated Search & Rescue instances.

Keywords

Configuration, heterogeneous fleet, CP, Search & Rescue.

1 Introduction

Aujourd'hui, les véhicules autonomes ou *robots* sont de plus en plus impliqués au sein de missions complexes (*p. ex.* surveillance de zones, livraison de colis, logistique d'entrepôts, etc.). Cette présence s'explique notamment par l'hétérogénéité de ces robots, provenant d'une part de leur disposition à accéder à des zones différentes via des routes aériennes, terrestres ou maritimes, et d'autre part de leur capacité à manier plusieurs charges utiles différentes, les rendant extrêmement pratiques et efficaces dans l'exécution de missions complexes. La performance et la robustesse de flottes constituées de robots sont des enjeux importants pour le succès d'une mission. Ainsi, le problème de la configuration d'une flotte de robots est essentiel, notamment au sein de missions pour lesquelles le budget alloué ou la durée d'exécution sont restreints. Par exemple, dans le cadre d'une mission *Recherche & Sauvetage*, le temps mis pour venir en aide à chacune des personnes impliquées dans la catastrophe est critique.

La configuration d'une flotte de robots peut se définir à plusieurs niveaux. Dans le cas où la flotte est déjà constituée, le premier niveau de configuration correspond à l'affectation de la succession des tâches à effectuer pour chacun des robots. Il s'agit d'un problème d'optimisation nommé *problème de configuration du plan* (PCP). C'est à ce niveau que la performance et la robustesse de la flotte sont évaluées. Cependant, lorsque les missions nécessitent de traiter un nombre croissant de tâches ou lorsque des robots sont devenus obsolètes, la composition de la flotte doit être remise en question. Ces considérations sont prises en compte par le *problème de configuration de la flotte* (FCP), qui inclut le dimensionnement et l'organisation de la flotte de robots pour lui permettre de s'adapter de la meilleure façon possible aux différentes tailles et complexités de mission. Enfin, il peut être nécessaire d'acquérir de nouvelles compétences afin d'effectuer des tâches plus variées. Le *problème de configuration des agents* (ACP) consiste à personnaliser chaque robot en choisissant ses équipements dans un catalogue, lui conférant ainsi des compétences propres.

Ces trois problèmes de configuration, *problème de configuration du plan* (PCP), *problème de configuration de la flotte* (FCP) et *problème de configuration des agents* (ACP), sont

interconnectés, c'est-à-dire que chaque décision prise dans le cadre de l'étude de l'un de ces problèmes a un impact direct sur l'ensemble des autres décisions. Par exemple, une flotte de robots mal dimensionnée peut être à l'origine de l'échec d'une mission et, à l'inverse, une mission peut imposer des configurations de flotte ou de robots spécifiques. Ainsi, afin d'éviter des cycles itératifs de configuration pour atteindre les objectifs de performance et de robustesse désirés, une approche holistique doit être envisagée. L'objectif de cet article est précisément d'étudier ces problèmes de configuration par une telle approche. Les contributions de cet article sont les suivantes.

1. le problème incluant les trois niveaux de configuration, nommé *problème global de configuration* (GCP), est formellement défini ;
2. un modèle de programmation par contraintes (PPC) est établi pour ce problème ;
3. de nouvelles instances académiques sont proposées ;
4. l'approche PPC est évaluée sur les nouvelles instances avec le solveur IBM ILOG CP Optimizer [10].

La structure de l'article est la suivante. Un état de l'art sur les problèmes de configuration de flottes d'agents hétérogènes est établi dans la section 2. Nous décrivons le problème global dans la section 3, puis nous présentons le modèle PPC dans la section 4. Nos expérimentations et les résultats associés sont établis dans la section 5. Enfin, une conclusion et une discussion des perspectives sont proposées dans la section 6.

2 État de l'art

Cette section analyse les travaux connexes à nos problématiques de configuration.

2.1 Configuration du plan

Le problème PCP présente des aspects liés aux problèmes de tournées de véhicules (*vehicle routing*) et aux problèmes d'ordonnancement de tâches (*production scheduling*). Ces problèmes sont extensivement étudiés dans la littérature ; parmi les plus célèbres, nous pouvons nommer le Capacitated Vehicle Routing Problem with Time-Windows [19] et le Job Shop Problem [9].

Le problème PCP présente également des contraintes liées aux ressources, c'est-à-dire que certaines tâches ne peuvent être effectuées que par des ressources spécialisées. Ainsi, le problème PCP fait partie d'une classe de problèmes nommée Resource-Constrained Routing and Scheduling Problems, présentée dans une revue de littérature de Paraskevopoulos *et al.* [13]. Dans cette revue, deux variantes principales de ces problèmes sont étudiées : le Skill VRP [1] et le Technician Routing and Scheduling Problem [14]. Le Skill VRP consiste à minimiser les coûts de trajets sous contraintes de qualification de techniciens (certaines tâches nécessitent un certain niveau de qualification). Les modèles proposés pour ce problème sont des programmes linéaires en nombres entiers. Ce problème est une spécialisation du Site-Dependant Vehicle Routing Problem (SDVRP) étudié par Cordeau et

Laporte dans [3]. Ce problème prend en compte de façon plus générale des contraintes de compatibilités entre les véhicules et les tâches à effectuer. Les instances de Cordeau et de Laporte ont été résolues par une recherche taboue.

Le Technician Routing and Scheduling Problem (TRSP) considère les visites d'un technicien dans des fenêtres de temps, nécessitant à la fois des compétences spécifiques mais également des outils et des pièces détachées. Les instances de TRSP sont résolues par une heuristique basée sur la méthode Adaptative Large Neighborhood Search (ALNS) développée par Ropke et Pisinger dans [17].

2.2 Couplage avec les décisions d'investissement

De nombreuses applications nécessitent de remettre en cause le dimensionnement de la flotte en vue de certaines missions. Une telle application peut être celle d'une entreprise de transport public souhaitant renouveler sa flotte de véhicules, pour des considérations économiques ou écologiques. Yildirim et Yildiz [22] proposent une méthode basée sur la génération de colonnes pour adresser le problème d'optimisation d'une flotte de bus et tirer parti des nouvelles technologies de recharge des batteries de bus électriques. Lemme *et al.* [11] évaluent par une programmation linéaire en nombres entiers le nombre de véhicules électriques à considérer dans une flotte d'un système de partage de voiture selon différents scénarios (*p. ex.* avec une contrainte plus ou moins forte sur les émissions de gaz à effet de serre selon la politique adoptée). Sayarshad et Tavakkoli-Moghaddam [18] proposent un modèle d'optimisation bi-niveau afin de dimensionner une flotte de wagons sous incertitudes de demandes. Ce modèle stochastique est résolu par la méthode de recuit simulé. Gamst et Pisinger [6] proposent différents types de décisions d'investissement pour le TRSP. Nous pouvons notamment citer l'évaluation de l'entraînement d'un technicien, le faisant ainsi monter en compétence, ou alors l'emploi d'un nouveau technicien. Une génération de colonnes est utilisée pour résoudre ce problème, en incluant les décisions d'investissement dans le problème maître, et les décisions d'affectation de tâches dans le sous-problème.

2.3 Application aux flottes de robots

Les trois niveaux de configuration pour une flotte de robots hétérogènes ont été identifiés par Pouré *et al.* [15]. Dans cet article, les auteurs proposent une formalisation des entrées et des sorties des trois problèmes de configuration. La résolution de ces problèmes est envisagée à la fois de façon séquentielle (la sortie de ACP vue comme entrée de FCP, et la sortie de FCP vue comme entrée de PCP) et itérative (l'évaluation de la performance obtenue dans PCP impacte les décisions prises dans ACP et FCP). Gigante *et al.* [8] proposent une approche basée sur la théorie des jeux pour configurer une flotte de drones aériens dans le but de réaliser des tâches d'observation de zones fixes. Cet article traite de nombreuses contraintes opérationnelles propres aux missions effectuées par des robots, telles que les contraintes de charge maximale, d'endurance (autonomie énergétique

du véhicule) ou encore de traficabilités (accessibilité à certaines zones). Notre contribution vis-à-vis de cet article est la proposition d'un modèle de programmation par contraintes pour résoudre le problème.

3 Description du problème

Dans cette section, nous décrivons les entrées et les sorties du problème GCP, ainsi que les postulats établis. Un exemple *fil rouge* d'une mission *Recherche & Sauvetage* est également introduit pour illustrer chacun des concepts.

3.1 Définitions et notations des entrées

Description de l'instance. Une instance de GCP est un tuple $(\mathcal{P}, \mathcal{E}, \mathcal{T}, \Pi, H, C, b)$ tel que :

- $\mathcal{P}$ est l'ensemble des plateformes disponibles dans le stock. Une plateforme est un robot non-équipé;
- $\mathcal{E}$ est l'ensemble des équipements disponibles dans le stock. Un équipement est une charge utile fournissant une compétence spécifique;
- $\mathcal{T}$ est l'ensemble des tâches devant être réalisées pour réussir la mission. Chaque tâche requiert une compétence spécifique fournie par un équipement;
- $\Pi \subset \mathcal{T} \times \mathcal{T}$ est l'ensemble des précédences à respecter entre les tâches. Une précédence est une relation entre deux tâches, imposant la première tâche à être effectuée avant la seconde tâche;
- $H \in \mathbb{N}^*$ est l'horizon temporel considéré pour la mission;
- $C \in \mathbb{R}^+$ est le budget maximal alloué à la mission;
- b est la base, i.e. le point de départ et de retour pour les robots de la flotte.

Dans les paragraphes suivants, nous décrivons les caractéristiques plus précises de chacun de ces éléments.

Description des plateformes. Pour chaque plateforme $p \in \mathcal{P}$:

- $c_p \in \mathbb{R}^+$ est son coût;
- $W_p \in \mathbb{R}^+$ est la charge maximale qu'elle peut embarquer;
- $d_p : \mathcal{T}_p \cup \{b\} \times \mathcal{T}_p \cup \{b\} \rightarrow \mathbb{N}$ est la fonction qui renvoie la durée nécessaire à la plateforme p pour se déplacer d'une opération à une autre. Une opération est soit une tâche atteignable par la plateforme p , soit un départ ou un retour à sa base;
- $\mathcal{T}_p \subseteq \mathcal{T}$ est l'ensemble des tâches atteignables par la plateforme p .

Description des équipements. Pour chaque équipement $e \in \mathcal{E}$:

- $c_e \in \mathbb{R}^+$ est son coût;
- $w_e \in \mathbb{R}^+$ est son poids;
- $\mathcal{P}_e \subseteq \mathcal{P}$ est l'ensemble des plateformes compatibles avec l'équipement e ;
- $\mathcal{E}_e \subseteq \mathcal{E}$ est l'ensemble des équipements compatibles avec l'équipement e .

Description des tâches. Pour chaque tâche $t \in \mathcal{T}$:

- $\delta_t \in \mathbb{N}^*$ est sa durée d'exécution;
- $\mathcal{E}_t \subseteq \mathcal{E}$ est l'ensemble des équipements possédant la compétence requise par la tâche t .

p	c_p	W_p	d_p	$\mathcal{T}_p$
	30	2	$20 \cdot f$	$\mathcal{T}$
	100	10	$50 \cdot f$	$\mathcal{T}$

TABLE 1 – Caractéristiques des plateformes de l'instance *fil rouge*. Les valeurs de la fonction f sont définies dans la table 2.

f				
	0	1	1	1
	1	0	0	1
	1	0	0	1
	1	1	1	0

TABLE 2 – Valeurs de la fonction de distance f

Description des précédences. Pour chaque précédence $(t_1, t_2) \in \Pi$, t_1 représente la tâche devant être effectuée avant la tâche t_2 .

Exemple 1 (instance *fil rouge*) Nous considérons la mission *Recherche & Sauvetage* suivante.

- l'ensemble des plateformes disponibles $\mathcal{P}$ est constitué d'un drone  et d'un rover . Leurs caractéristiques sont présentées dans la table 1. Ces plateformes peuvent accéder aux mêmes tâches mais se déplacent à une vitesse différente. Ainsi, les valeurs des fonctions d_{drone} et d_{rover} sont déterminées à partir d'une fonction de distance $f : \mathcal{T} \cup \{b\} \times \mathcal{T} \cup \{b\} \rightarrow \mathbb{N}$ définie dans la table 2;
- l'ensemble des équipements disponibles $\mathcal{E}$ est constitué d'une caméra , d'un bras mécanique , d'un parachute  et d'une remorque . Leurs caractéristiques sont présentées dans la table 3;
- l'ensemble des tâches à réaliser $\mathcal{T}$ est constitué d'une observation , d'une réparation  et d'une livraison . Leurs caractéristiques sont présentées dans la table 4;
- l'ensemble des précédences Π est constitué d'une unique relation imposant que la tâche d'observation  soit effectuée avant la tâche de réparation . Ainsi, $\Pi = \{(\text{caméra}, \text{réparation})\}$;

e	c_e	w_e	$\mathcal{P}_e$	$\mathcal{E}_e$
	15	1	$\mathcal{P}$	$\mathcal{E}$
	25	3	$\{\text{rover}\}$	$\mathcal{E}$
	1	1	$\{\text{drone}\}$	$\mathcal{E}$
	10	6	$\{\text{rover}\}$	$\mathcal{E}$

TABLE 3 – Caractéristiques des équipements de l'instance *fil rouge*

t	δ_t	$\mathcal{E}_t$
	30	$\{\text{camera}\}$
	60	$\{\text{tool}\}$
	12	$\{\text{drone}, \text{box}\}$

TABLE 4 – Caractéristiques des tâches de l'instance *fil rouge*

- $H = 360$;
- $C = 180$;
- la base b est l'entrepôt .

3.2 Définitions et notations des sorties

Une solution pour une instance de GCP est un couple $(\mathcal{A}, \mathcal{R})$, avec $\mathcal{A}$ représentant l'ensemble des agents de la flotte configurée pour la mission et $\mathcal{R}$ l'ensemble des routes de chacun des agents.

Description des agents. Pour chaque agent $a \in \mathcal{A}$:

- $p_a \in \mathcal{P}$ est la plateforme utilisée par l'agent a ;
- $\mathcal{E}_a \subseteq \mathcal{E}$ est l'ensemble des équipements associés à l'agent a .

Description des routes. Pour chaque route $r \in \mathcal{R}$:

- $\text{agent}_{r_1} \in \mathcal{A}$ est l'agent utilisé pour parcourir la route r ;
- $\mathbf{V}_r$ est la séquence des visites effectuées sur la route r . Pour tout $i \in \llbracket 1, |\mathbf{V}_r| \rrbracket$ ¹, une visite $v_r^i \in \mathbf{V}_r$ est un couple (o_r^i, h_r^i) tel que :
 - o_r^i est l'opération réalisée pendant la visite v_r^i vérifiant :

$$\begin{cases} o_r^i = b & \text{si } i = 1 \text{ ou } i = |\mathbf{V}_r| \\ o_r^i \in \mathcal{T} & \text{si } i \in \llbracket 2, |\mathbf{V}_r| - 1 \rrbracket \end{cases};$$

- $h_r^i \in \llbracket 0, H \rrbracket$ est la date à laquelle la visite v_r^i est effectuée.

Une solution est *faisable* si les agents, formés d'associations valides entre les plateformes et les équipements, sont capables de réaliser l'intégralité des tâches dans le respect des contraintes opérationnelles (budget, précédences, compatibilités, etc.).

Une solution est *optimale* si elle est faisable et si elle minimise le *makespan*, correspondant à la date de retour à la base du dernier agent.

Exemple 2 (solutions pour l'instance *fil rouge*) Une solution faisable pour l'instance *fil rouge* est la suivante :

- $\mathcal{A} = \{a_1, a_2\}$ avec :
 - $p_{a_1} = \text{drone}$ et $p_{a_2} = \text{robot}$;
 - $\mathcal{E}_{a_1} = \{\text{camera}\}$ et $\mathcal{E}_{a_2} = \{\text{tool}, \text{box}\}$;
- $\mathcal{R} = \{r_1, r_2\}$ avec :
 - $\text{agent}_{r_1} = a_1$ et $\text{agent}_{r_2} = a_2$;
 - $\mathbf{V}_{r_1} = [(\text{entrepôt}, 0), (\text{camera}, 20), (\text{entrepôt}, 70)]$;
 - $\mathbf{V}_{r_2} = [(\text{entrepôt}, 0), (\text{tool}, 50), (\text{box}, 160), (\text{entrepôt}, 222)]$.

1. La notation $\llbracket \cdot, \cdot \rrbracket$ désigne l'ensemble des entiers entre deux bornes incluses.

Les dates auxquelles sont effectuées les tâches sont calculées au plus tôt. Ainsi, l'agent a_1 commence l'observation à la date 20, la termine à la date 50 et retourne à l'entrepôt à la date 70. L'agent a_2 commence la réparation à la date 50 (la contrainte de précédence étant bien respectée), la termine à la date 110, puis enchaîne sur la livraison à la date 160, qui se termine à la date 172. La date de retour à l'entrepôt de l'agent a_2 est 222, qui est donc la valeur du *makespan* associé à cette solution. Le budget dépensé est 180, ce qui satisfait la contrainte de budget maximal. Cette solution n'est pas optimale, il est en effet plus performant d'affecter la livraison à l'agent a_1 car le drone est plus rapide. La solution optimale pour l'instance *fil rouge* est la suivante :

- $\mathcal{A} = \{a_1, a_2\}$ avec :
 - $p_{a_1} = \text{drone}$ et $p_{a_2} = \text{robot}$;
 - $\mathcal{E}_{a_1} = \{\text{camera}, \text{box}\}$ et $\mathcal{E}_{a_2} = \{\text{tool}\}$;
- $\mathcal{R} = \{r_1, r_2\}$ avec :
 - $\text{agent}_{r_1} = a_1$ et $\text{agent}_{r_2} = a_2$;
 - $\mathbf{V}_{r_1} = [(\text{entrepôt}, 0), (\text{camera}, 20), (\text{box}, 70), (\text{entrepôt}, 102)]$;
 - $\mathbf{V}_{r_2} = [(\text{entrepôt}, 0), (\text{tool}, 50), (\text{entrepôt}, 160)]$.

Il est important de noter que l'ordre des opérations effectuées par l'agent a_1 est important : si la livraison est effectuée avant l'observation , alors celle-ci ne sera pas effectuée avant la date 82, ce qui dégrade le *makespan* de 32 unités de temps en raison de la contrainte de précédence (l'agent a_2 doit attendre la fin de l'observation réalisée par l'agent a_1 avant de commencer la réparation .

3.3 Postulats

Les postulats établis pour l'étude de ce problème sont les suivants.

1. les relations de compatibilité entre les équipements doivent être cohérentes entre elles, c'est-à-dire que $\forall (e_1, e_2) \in \mathcal{E}^2$:

$$e_2 \in \mathcal{E}_{e_1} \iff e_1 \in \mathcal{E}_{e_2};$$

2. les précédences entre les tâches forment un graphe dirigé acyclique ;
3. les agents constitués sont libres de se déplacer sans risque de conflits avec les autres agents. Ainsi, dans le cadre de la minimisation de la durée de la mission, les agents ont toujours la possibilité d'emprunter le plus court chemin entre les tâches ;
4. les agents présents dans la solution sont utiles, c'est-à-dire que chaque agent doit effectuer au moins une tâche ;
5. les agents ne peuvent pas coopérer pour la réalisation des tâches. Les tâches n'ont donc besoin que d'un seul agent pour être réalisées et aucune synchronisation d'agents n'est à considérer ;
6. la mission est statique, c'est-à-dire que l'environnement et les tâches sont parfaitement connus à l'avance ;
7. les plateformes et les équipements sont sans défauts. La robustesse de la configuration n'est donc pas évaluée dans cet article.

4 Modèle PPC

Dans cet article, nous utilisons la formulation à base de variables de type *intervalle* du langage de modélisation OPL [20]. Une variable d'intervalle permet de représenter une tâche ou une activité par sa date de début, sa date de fin, sa durée et sa présence obligatoire ou facultative dans la solution. Ces informations sont respectivement accessibles via les opérateurs `startOf`, `endOf`, `lengthOf` et `presenceOf`. Il est possible de préciser la valeur de chaque opérateur lorsque l'intervalle optionnel sur lequel il s'applique est absent. Par exemple, `startOf(itv, 0)` renvoie la date de début de l'intervalle `itv` si celui-ci est présent, et 0 s'il est absent.

4.1 Variables de décision

Pour chaque tâche $t \in \mathcal{T}$, nous considérons la variable d'intervalle $\text{itv}_t \in [0, H]$ représentant l'exécution de la tâche t . Sa présence est obligatoire et sa durée est constante et égale à δ_t . Pour chaque plateforme $p \in \mathcal{P}$ et chaque tâche $t \in \mathcal{T}_p$, nous considérons la variable d'intervalle $\text{itv}_{t,p} \in [0, H]$ représentant l'exécution de la tâche t par la plateforme p . Sa présence est facultative et sa durée est constante et égale à δ_t . Pour chaque équipement $e \in \mathcal{E}$, nous considérons la variable entière $\text{pf}_e \in \mathcal{P}_e \cup \{p^\emptyset\}$ représentant la plateforme associée à l'équipement e . La notation $p^\emptyset$ représente la plateforme fictive des équipements inutilisés. Pour chaque plateforme $p \in \mathcal{P}$, nous considérons les variables suivantes.

- $\text{itv}_p^\alpha \in [0, H]$ est la variable d'intervalle représentant la tâche de départ de la base. Sa présence est facultative et sa durée est nulle;
- $\text{itv}_p^\Omega \in [0, H]$ est la variable d'intervalle représentant la tâche d'arrivée à la base. Sa présence est facultative et sa durée est nulle.

4.2 Expressions intermédiaires

Pour chaque plateforme $p \in \mathcal{P}$, nous considérons la variable booléenne $\mathbf{u}_p \in \{0, 1\}$ indiquant si la plateforme p est utilisée. Cette variable est définie dans l'équation (1).

$$\mathbf{u}_p = \left(\left[\sum_{t \in \mathcal{T}_p} \text{presenceOf}(\text{itv}_{t,p}) \right] > 0 \right) \quad (1)$$

Pour chaque équipement $e \in \mathcal{E}$, nous considérons la variable booléenne $\mathbf{y}_e \in \{0, 1\}$ indiquant si l'équipement e est utilisé. Cette variable est définie dans l'équation (2).

$$\mathbf{y}_e = (\text{pf}_e \neq p^\emptyset) \quad (2)$$

Pour chaque équipement $e \in \mathcal{E}$ et pour chaque plateforme $p \in \mathcal{P}$, nous considérons la variable booléenne $\mathbf{x}_{e,p} \in \{0, 1\}$ indiquant si l'équipement e est associé à la plateforme p . Cette variable est définie dans l'équation (3).

$$\mathbf{x}_{e,p} = (\text{pf}_e = p) \quad (3)$$

Il est important de noter que les définitions de $\mathbf{u}_p$, $\mathbf{y}_e$ et $\mathbf{x}_{e,p}$ sont permises grâce au mécanisme de réification de contraintes, qui associe une variable booléenne à la satisfaction d'une contrainte.

4.3 Contraintes

Grâce aux notations définies précédemment, nous définissons les contraintes de notre modèle.

La contrainte de respect du budget maximal est exprimée dans l'équation (4).

$$\sum_{p \in \mathcal{P}} c_p \mathbf{u}_p + \sum_{e \in \mathcal{E}} c_e \mathbf{y}_e \leq C \quad (4)$$

La contrainte de charge maximale des plateformes est exprimée dans l'équation (5). Le respect des durées de déplacement entre les tâches d'une même plateforme est garanti par l'équation (6) via l'opérateur `noOverlap` du langage OPL. Cet opérateur prend en entrée un ensemble d'intervalles pour lesquels il impose le non-chevauchement, ainsi que la fonction des temps de transition entre les tâches qui doivent être respectés.

$$\forall p \in \mathcal{P}, \quad \sum_{e \in \mathcal{E}} w_e \mathbf{x}_{e,p} \leq W_p \quad (5)$$

$$\forall p \in \mathcal{P}, \quad \text{noOverlap}(\{\text{itv}_{t,p}\}_{t \in \mathcal{T}_p}, d_p) \quad (6)$$

Les contraintes (7) empêchent deux équipements incompatibles d'être associés à la même plateforme.

$$\forall e_1 \in \mathcal{E}, \forall e_2 \in \mathcal{E} \setminus \mathcal{E}_{e_1}, \quad \mathbf{y}_{e_1} \wedge \mathbf{y}_{e_2} \rightarrow (\text{pf}_{e_1} \neq \text{pf}_{e_2}) \quad (7)$$

Les contraintes d'association d'un unique intervalle optionnel à chaque tâche $t \in \mathcal{T}$ sont exprimées grâce à l'opérateur `alternative` du langage OPL dans l'équation (8). Cet opérateur prend en entrée un intervalle et un ensemble d'intervalles optionnels. Si l'intervalle donné en entrée est présent, alors exactement un des intervalles optionnels donnés en entrée est présent et leurs dates de débuts, dates de fins et durées sont égales.

$$\forall t \in \mathcal{T} \quad \text{alternative}(\text{itv}_t, \{\text{itv}_{t,p}\}_{p \in \mathcal{P}_t}) \quad (8)$$

Les contraintes de précédence entre les tâches sont gérées par l'opérateur `endBeforeStart` du langage OPL dans l'équation (9). Cet opérateur prend en entrée deux intervalles et impose que le premier intervalle se termine avant le début du second intervalle, avec un délai éventuel à respecter. Cette contrainte est levée si l'un des deux intervalles est absent.

$$\forall (t_1, t_2) \in \Pi \quad \text{endBeforeStart}(\text{itv}_{t_1}, \text{itv}_{t_2}) \quad (9)$$

Les contraintes (10) et (11) assurent que toutes les tâches d'une plateforme s'effectuent après son départ de la base.

$$\forall p \in \mathcal{P}, \forall t \in \mathcal{T}_p, \quad \text{endBeforeStart}(\text{itv}_p^\alpha, \text{itv}_{t,p}, d_p(b, t)) \quad (10)$$

$$\forall p \in \mathcal{P}, \quad \text{presenceOf}(\text{itv}_p^\alpha) \geq \mathbf{u}_p \quad (11)$$

Les contraintes (12) et (13) assurent que toutes les tâches d'une plateforme s'effectuent avant son retour à la base.

$$\forall p \in \mathcal{P}, \forall t \in \mathcal{T}_p, \quad \text{endBeforeStart}(\text{itv}_{t,p}, \text{itv}_p^\Omega, d_p(t, b)) \quad (12)$$

$$\forall p \in \mathcal{P}, \quad \text{presenceOf}(\text{itv}_p^\Omega) \geq \mathbf{u}_p \quad (13)$$

Enfin, les contraintes de compatibilité entre les agents et les tâches, issues des compétences apportées par les équipements et des requêtes des tâches, sont exprimées dans l'équation (14).

$$\forall p \in \mathcal{P}, \forall t \in \mathcal{T}_p, \sum_{e \in \mathcal{E}_t} x_{e,p} \geq \text{presenceOf}(\text{itv}_{t,p}) \quad (14)$$

4.4 Formulation du problème

Le critère retenu est la minimisation du *makespan* exprimé dans l'équation (15). La formulation du problème est la suivante :

$$\begin{aligned} & \text{minimize} && \max_{p \in \mathcal{P}} \text{endOf}(\text{itv}_p^\Omega) && (15) \\ & \text{s.t.} && && (4) - (14) \end{aligned}$$

5 Expérimentations

5.1 Génération d'instances

Nous avons généré un ensemble d'instances afin de tester notre approche. Dans l'instance *fil rouge* présentée dans la section 3.1, tous les éléments du stock ont des caractéristiques différentes. Cependant, dans les instances de nos expérimentations, certains éléments sont identiques et ont donc les mêmes caractéristiques. Dans cette section, nous utilisons une notation unique pour désigner les éléments d'un même type (*p. ex.*  représente un drone quelconque du stock disponible). De façon similaire, nous utilisons une notation unique pour désigner les tâches du même type, c'est-à-dire de même durée et nécessitant les mêmes compétences.

Méthodologie. Pour générer ces instances, nous avons créé un squelette d'instance générique pour lequel certains paramètres sont fixés, puis nous avons ajouté de la variabilité à ce squelette afin d'enrichir notre jeu de données.

Paramètres généraux. Les principales caractéristiques des instances générées sont les suivantes.

- les caractéristiques des plateformes sont présentées dans la table 5. Pour chaque plateforme $p \in \mathcal{P}$, les valeurs de la fonction d_p sont déterminées à partir d'une fonction de distance $f_p : \mathcal{T}_p \cup \{b\} \times \mathcal{T}_p \cup \{b\} \rightarrow \mathbb{R}^+$, qui représente la distance que chaque plateforme doit parcourir pour passer d'une tâche à une autre, et dont le calcul est détaillé dans la suite de cette section²;
- les caractéristiques des équipements sont présentées dans la table 6. Par rapport à l'instance *fil rouge*, un nouveau type d'équipement est disponible dans le stock : l'émetteur ;
- les caractéristiques des tâches sont présentées dans la table 7. Par rapport à l'instance *fil rouge*, un nouveau type de tâche peut être réalisé : l'activation de balises ;
- l'ensemble des précédences est vide;

2. Les valeurs de d_p sont arrondies à l'entier supérieur.

- le budget maximal alloué est choisi dans $C^{\min} \times \{1, 2, 3, 4\}$ où $C^{\min}$ représente le budget minimal pour effectuer la mission. Notons que ce budget minimal est obtenu par énumération des associations entre les types de plateformes et les types d'équipements, ce qui est permis par la faible combinatoire de notre étude sur ces grandeurs.
- la base est l'entrepôt .

p	c_p	W_p	d_p	$\mathcal{T}_p$
	3000	2	$20 \cdot f_{\text{drone}}$	$\mathcal{T}$
	10000	10	$50 \cdot f_{\text{robot}}$	$\mathcal{T}$

TABLE 5 – Caractéristiques des plateformes des instances générées

e	c_e	w_e	$\mathcal{P}_e$	$\mathcal{E}_e$
	1500	1	$\mathcal{P}$	$\mathcal{E}$
	300	1	$\{\text{drone}\}$	$\mathcal{E} \setminus \{\text{robot}\}$
	2500	3	$\{\text{robot}\}$	$\mathcal{E}$
	100	1	$\{\text{drone}\}$	$\mathcal{E} \setminus \{\text{antenna}\}$
	1000	6	$\{\text{robot}\}$	$\mathcal{E}$

TABLE 6 – Caractéristiques des équipements des instances générées

t	δ_t	$\mathcal{E}_t$
	300	$\{\text{camera}\}$
	600	$\{\text{antenna}\}$
	600	$\{\text{wrench}\}$
	120	$\{\text{robot}, \text{warehouse}\}$

TABLE 7 – Caractéristiques des tâches des instances générées

Nombre de tâches. Le nombre de tâches à effectuer est choisi dans $\{4, 8, 20, 48\}$. Les proportions des types de tâches sont présentées dans la colonne "Proportion" de la table 8.

Dimensionnement du stock. Le stock de plateformes est considéré illimité. En pratique, une plateforme par tâche compatible est considérée dans le stock disponible. La distribution des équipements par rapport aux compétences requises par les tâches est choisie dans $\{\text{same}, \text{opposite}\}$. La distribution *same* correspond à un nombre d'équipements en adéquation avec la proportion de chaque tâche, alors que la distribution *opposite* présente un nombre d'équipements en excès pour les tâches rares, et en défaut pour les tâches répandues. Les stocks d'équipements, définis en fonction de leur distribution, sont présentés dans la table 8.

t	Proportion	e	Stock (<i>same</i>)	Stock (<i>opp.</i>)
	50%		9	1
	15%		5	9
	15%		5	9
	20%		3	3
			3	3

TABLE 8 – Proportion des types de tâches et stock d'équipements dans les instances générées

Localisation des opérations. Les tâches à réaliser et la base sont localisées sur un graphe. L'ensemble des nœuds de ce graphe est extrait des 51 nœuds de l'instance CMT01 proposée dans le cadre de l'étude du Capacitated Vehicle Routing Problem (CVRP) dans [2]. Pour chaque instance, $|\mathcal{T}| + 1$ nœuds sont extraits afin de permettre la répartition des tâches et de la base sur des nœuds différents. La position de la base est choisie dans $\{side, center\}$. La position *side* correspond au nœud du graphe le plus éloigné de tous les autres, alors que la position *center* correspond au nœud le plus central. Les types de tâches sont répartis aléatoirement entre les nœuds restants.

Calcul des durées de déplacement pour . Nous souhaitons qu'un drone  puisse naviguer librement entre les nœuds en considérant le graphe complet. Ainsi, les valeurs de f_{drone} pour chaque couple d'opérations sont égales à la distance euclidienne entre les nœuds sur lesquels ces opérations sont localisées.

Calcul des durées de déplacement pour . Nous souhaitons également qu'un rover  ne puisse naviguer qu'entre les nœuds proches en considérant un graphe planaire et connexe. Ce graphe est obtenu en appliquant une triangulation de Delaunay [4] aux nœuds de chaque instance. Dans ce graphe, la distance entre deux nœuds adjacents est égale à la distance euclidienne. Pour obtenir la distance entre deux nœuds non adjacents, il faut déterminer le plus court chemin qui les relie. Un tel chemin existe car le graphe est connexe. Nous utilisons l'algorithme de Floyd-Warshall [5, 21] afin de calculer ces plus courts chemins et donc définir la fonction de distance f_{rover} .

Calcul de l'horizon. Il est nécessaire de définir l'horizon temporel H afin de borner les variables d'intervalle du modèle. Dans ce but, nous définissons dans l'équation (16) la durée maximale de parcours d'une arête du graphe par une plateforme quelconque :

$$D^{max} = \max_{p \in \mathcal{P}} d_p \quad (16)$$

Ainsi, en considérant une unique plateforme qui effectuerait l'intégralité des tâches et se déplacerait vers, depuis et entre chaque opération avec la durée D^{max} , nous obtenons une borne supérieure pour les variables d'intervalle, que nous

prenons comme définition de l'horizon dans l'équation (17).

$$H = \left(\sum_{t \in \mathcal{T}} \delta_t \right) + (|\mathcal{T}| + 1) D^{max} \quad (17)$$

Nommage des instances. La convention choisie pour les noms des instances est $t\{nTasks\}-c\{budget\}-\{distrib\}-\{base\}$ avec :

- $nTasks$ représente le nombre de tâches à effectuer : "4", "8", "20" ou "48";
- $budget$ représente le facteur associé au budget maximal alloué : "1", "2", "3" ou "4";
- $distrib$ représente la distribution des équipements : "same" ou "opp" (pour une distribution *opposite*);
- $base$ représente la position de la base : "side" ou "center".

Ainsi, un ensemble de $4 \times 4 \times 2 \times 2 = 64$ instances ont été générées à partir de ces paramètres variables. Ces instances sont disponibles dans [12].

5.2 Environnement de test

Toutes nos expérimentations ont été réalisées sous Windows Subsystem for Linux 2 (WSL2 version 2.2.4, distribution Ubuntu 24.04.1 LTS et architecture 64-bits) avec un processeur Intel Core i5-1345U, 6 cœurs (12 threads) et 8 Go de RAM. Le modèle a été écrit et testé avec Java 23 et IBM ILOG CP Optimizer 22.1.2.

5.3 Résultats

Le modèle défini dans la section 4 utilise le formalisme du langage OPL. De ce fait, aucune traduction des variables de décision, des contraintes ou des critères n'est requise pour l'utilisation de CP Optimizer. Chaque résolution est paramétrée avec une limite de temps égale à 60 secondes. Pour des raisons de place, nous ne présentons pas les résultats des 64 instances, uniquement les 16 instances avec la distribution des équipements *same* et la position de la base *side* (instances *same-side*), et les 16 instances avec la distribution des équipements *opposite* et la position de la base *center* (instances *opposite-center*). Les résultats des instances *same-side* sont détaillés dans la table 9. Nous observons que la résolution converge en quelques secondes seulement sur les instances allant jusqu'à 20 tâches. Pour les instances comprenant 48 tâches (et pour l'instance t20-c3-same-side), l'optimalité n'a pas pu être prouvée en 60 secondes et aucun gap d'optimalité n'a pas été obtenu, ce qui montre les limites de la modélisation actuelle. Une autre modélisation, non présentée dans cet article, a été développée afin de renforcer la confiance en ces résultats. Cette modélisation, basée sur des variables de décision ensemblistes, a été implémentée et testée avec le solveur Hexaly Optimizer [7]. Les valeurs de *makespan* obtenues avec Hexaly Optimizer 13.5 en 60 secondes sont identiques à celles présentées dans la table 9. Le choix de Hexaly Optimizer pour cette validation est justifié par ses excellentes performances pour la résolution de problèmes de tournées de véhicules.

Les résultats des instances *opposite-center* sont détaillés dans la table 10. Similairement aux résultats de la table 9,

Instance	Makespan	Opt.	Time (s)
t4-c1-same-side	4480	*	0.02
t4-c2-same-side	3524	*	0.03
t4-c3-same-side	3524	*	0.03
t4-c4-same-side	3524	*	0.03
t8-c1-same-side	4890	*	0.15
t8-c2-same-side	3814	*	0.07
t8-c3-same-side	3814	*	0.07
t8-c4-same-side	3814	*	0.16
t20-c1-same-side	7765	*	0.35
t20-c2-same-side	5459	*	1.34
t20-c3-same-side	4654		60.00
t20-c4-same-side	4654	*	1.05
t48-c1-same-side	14838		60.00
t48-c2-same-side	10389		60.00
t48-c3-same-side	7426		60.00
t48-c4-same-side	6606		60.00

TABLE 9 – Valeurs de *makespan* pour les instances *same-side*. Un astérisque (*) indique une preuve d'optimalité.

la résolution converge rapidement sur les instances de petite taille. Nous observons cependant deux différences notables :

- Pour les instances de petite taille, les valeurs optimales de *makespan* sont plus basses dans la table 10 que dans la table 9. En effet, la base étant sur une position centrale, les tâches sont moins éloignées de la base et peuvent donc être effectuées plus rapidement par les agents lorsque le budget le permet.
- Pour les instances de grande taille, les valeurs optimales de *makespan* sont plus élevées dans la table 10 que dans la table 9. En effet, une seule caméra 📷 est disponible pour effectuer la totalité des observations 📷, ce qui limite l'amélioration du *makespan* même lorsque le budget alloué est élevé.

Focus sur l'instance t20-c1-same-side. Dans l'instance t20-c1-same-side, les paramètres sélectionnés sont les suivants.

- 20 tâches doivent être réalisées ;
- le budget est le plus petit possible ;
- les distributions des tâches et des équipements sont en adéquation ;
- la base est excentrée.

Nous représentons sur la figure 1 la carte associée à la solution pour cette instance. Sur cette carte, dont les axes représentent les coordonnées des nœuds associés aux tâches, nous avons représenté les tâches à effectuer et les routes empruntées par les agents. La base est située en haut à droite de la carte aux coordonnées (62, 63). Les bandes grises reliant les tâches correspondent au graphe planaire d'un rover 🚚. Il est aisé de voir que cette configuration respecte la contrainte de budget minimal. En effet, seule une instance de chaque type de plateforme est utilisée :

- l'agent "Drone 1" est un drone 🚁 équipé d'une caméra 📷 et d'un émetteur 📡 ;
- l'agent "Rover 1" est un rover 🚚 équipé d'un bras

Instance	Makespan	Opt.	Time (s)
t4-c1-opp-center	2553	*	0.02
t4-c2-opp-center	2250	*	0.03
t4-c3-opp-center	2250	*	0.06
t4-c4-opp-center	2250	*	0.04
t8-c1-opp-center	4557	*	0.10
t8-c2-opp-center	2879	*	1.52
t8-c3-opp-center	2879	*	1.63
t8-c4-opp-center	2879	*	1.86
t20-c1-opp-center	7645	*	1.10
t20-c2-opp-center	4909		60.00
t20-c3-opp-center	4909		60.00
t20-c4-opp-center	4909		60.00
t48-c1-opp-center	14851		60.00
t48-c2-opp-center	10093		60.00
t48-c3-opp-center	10093		60.00
t48-c4-opp-center	10093		60.00

TABLE 10 – Valeurs de *makespan* pour les instances *opposite-center*. Un astérisque (*) indique une preuve d'optimalité.

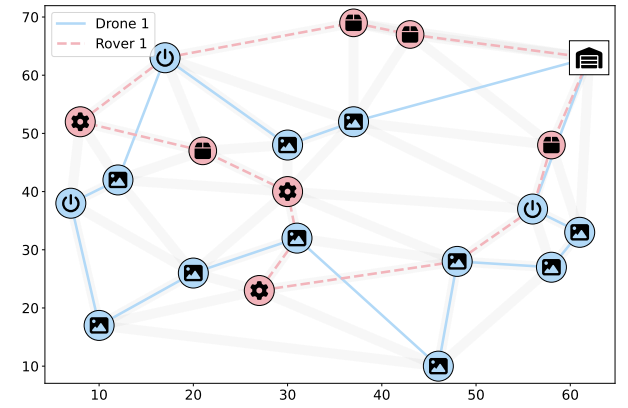


FIGURE 1 – Carte de la solution pour l'instance t20-c1-same-side

mécanique 🛠️ et d'une remorque 🚚.

Par ailleurs, toujours sous contrainte de budget minimal, il n'est pas possible d'écarter définitivement un type de plateforme. En effet, puisqu'un drone 🚁 ne peut pas être équipé d'un émetteur 📡 et d'un parachute 🪂 simultanément pour des raisons de compatibilité, alors il est nécessaire d'utiliser un rover 🚚 afin d'effectuer la livraison 📦.

Le diagramme de Gantt de la solution pour cette même instance est représenté sur la figure 2. Sur ce diagramme, on observe que la durée planifiée la plus longue correspond à la route du rover 🚚. Ce résultat est attendu puisque le rover 🚚 se déplace significativement plus lentement que le drone 🚁. On observe par ailleurs que les observations 📷 sont effectuées par le drone 🚁, ce qui est également attendu puisque celles-ci sont en plus grand nombre.

Notons ici que le drone 🚁 peut effectuer ses tâches moins optimalement et sans impact sur le *makespan*. Pour une

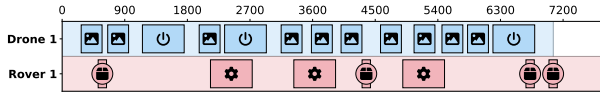


FIGURE 2 – Diagramme de Gantt de la solution pour l'instance t20-c1-same-side

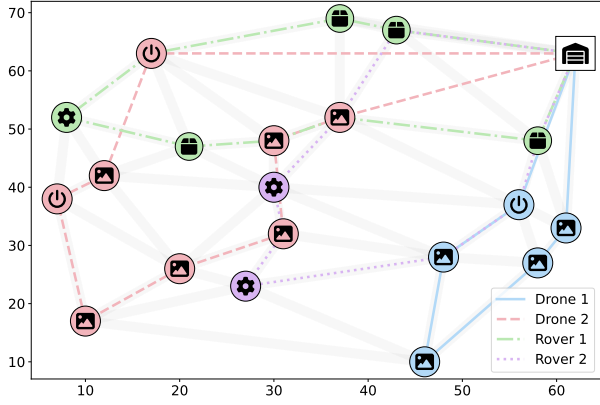


FIGURE 3 – Carte de la solution pour l'instance t20-c2-same-side

meilleure visualisation, nous avons décidé d'afficher dans nos figures les solutions optimales pour les critères de *makespan* et de durée des routes, traités dans un ordre lexicographique. Les performances de cette nouvelle modélisation ne sont pas abordées dans cet article.

Focus sur l'instance t20-c2-same-side. Les résultats précédemment présentés se comparent naturellement à la situation pour laquelle le budget maximal autorisé est doublé. La carte associée à la solution pour l'instance t20-c2-same-side est représentée sur la figure 3. Dans cette solution, nous observons l'utilisation d'un drone et d'un rover supplémentaires pour effectuer l'ensemble des tâches. Le diagramme de Gantt associé à cette solution est représenté sur la figure 4. Nous observons que la valeur du *makespan* a diminué de 30%, ce qui montre l'impact du budget sur la performance de la mission.

Fronts de Pareto des solutions pour les instances t20. Les fronts de Pareto représentés sur la figure 5 montrent les résultats des instances pour lesquelles 20 tâches doivent

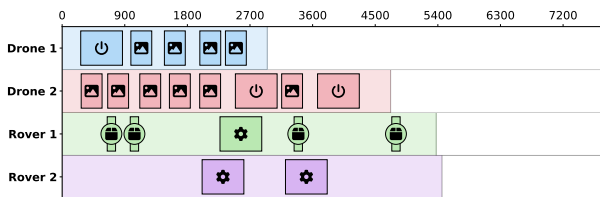


FIGURE 4 – Diagramme de Gantt de la solution pour l'instance t20-c2-same-side

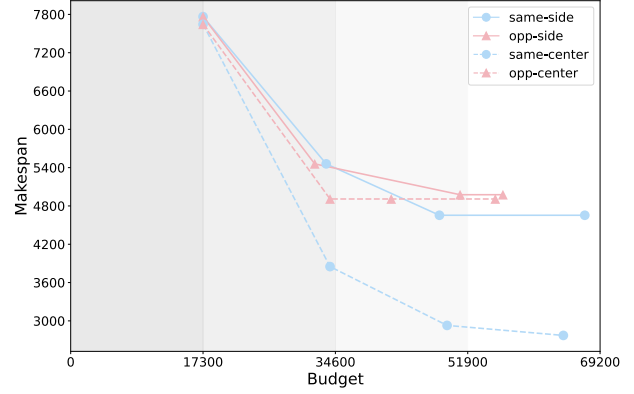


FIGURE 5 – Fronts de Pareto des solutions pour les instances t20

être réalisées (instances t20). Chaque front de Pareto est composé de 4 points représentant chaque solution par son budget sur l'axe des abscisses et la valeur du *makespan* sur l'axe des ordonnées. Les bandes grises représentent le budget maximal alloué, la contrainte étant progressivement relâchée de la gauche vers la droite.

Chaque front montre que la performance de la mission est fortement dégradée (*i.e.* la valeur du *makespan* croît) lorsque le budget alloué pour la mission se restreint. De plus, on observe qu'à partir d'un certain budget de mission, le *makespan* est difficilement améliorable par l'augmentation du budget de mission et donc par l'ajout potentiel de rovers ou de drones supplémentaires.

Nous observons qu'une distribution des équipements *same* permet d'atteindre des meilleures performances de mission qu'une distribution *opposite*. De plus, nous observons qu'une position de base *center* permet également d'obtenir de meilleures performances qu'une position de base *side*, ce qui est attendu.

6 Conclusion

Dans cet article, nous avons i) défini formellement le problème global incluant les trois niveaux de configuration de flottes hétérogènes, ii) établi un modèle de programmation par contraintes pour ce problème, et iii) testé et évalué ce modèle sur de nouvelles instances avec le solveur CP Optimizer.

L'un de nos travaux futurs est l'évaluation de notre modèle avec d'autres solveurs de programmation par contraintes tels que Choco-solver [16]. Ce travail pourra être complété par l'évaluation d'autres approches heuristiques (*p. ex.* recherche locale) ou exactes (*p. ex.* modèle MILP). Nous envisageons également d'enrichir le modèle afin de traiter des scénarios réalistes lors d'expérimentations in situ. Enfin, nous remettons en cause chacun des postulats énoncés dans la section 3.3, dans le but d'améliorer la pertinence de notre problème face aux enjeux réels de configuration.

Remerciements

Les travaux menés dans la thèse de Lucas Ligny sont financés par l'Agence de l'innovation de défense (AID) du Ministère des Armées.

Références

- [1] Paola Cappanera, Luís Gouveia, and Maria Grazia Scutellà. *The Skill Vehicle Routing Problem*, pages 354–364. Springer Berlin Heidelberg, 2011.
- [2] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. The vehicle routing problem. In Nicos Christofides, Aristide Mingozzi, Paolo Toth, and Claudio Sandi, editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, 1979.
- [3] Jean-François Cordeau and Gilbert Laporte. A tabu search algorithm for the site dependent vehicle routing problem with time windows. *INFOR : Information Systems and Operational Research*, 39(3) :292–298, August 2001.
- [4] Boris Delaunay. Sur la sphère vide. *Bulletin de l'Académie des Sciences de l'URSS, Classe des Sciences Mathématiques et Naturelles*, pages 793–800, 1934.
- [5] Robert W. Floyd. Algorithm 97 : Shortest path. *Commun. ACM*, 5(6) :345, June 1962.
- [6] Mette Gamst and David Pisinger. Decision support for the technician routing and scheduling problem. *Networks*, 83(1) :169–196, September 2023.
- [7] Frédéric Gardi, Thierry Benoist, Julien Darlay, Bertrand Estellon, and Romain Megel. *Mathematical Programming Solver Based on Local Search*. Wiley, June 2014.
- [8] Gabriella Gigante, Domenico Pascarella, Salvatore Luongo, Carlo Di Benedetto, Angela Vozella, and Giuseppe Persechino. Game-theoretic approach for the optimal configuration computing of an interoperable fleet of unmanned vehicles. *Expert Systems*, 35(5) :e12293, 2018. e12293 10.1111/exsy.12293.
- [9] Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9) :1563–1581, November 1966.
- [10] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vlířm. Ibm ilog cp optimizer for scheduling : 20+ years of scheduling with constraints at ibm/ilog. *Constraints*, 23(2) :210–250, March 2018.
- [11] Rafael F.F. Lemme, Edilson F. Arruda, and Laura Bahiense. Optimization model to assess electric vehicles as an alternative for fleet composition in station-based car sharing systems. *Transportation Research Part D : Transport and Environment*, 67 :173–196, 2019.
- [12] Lucas Ligny, Gauthier Picard, Stéphanie Roussel, and Elise Vareilles. Jeu de données pour le problème de configuration de flottes hétérogènes. <https://doi.org/10.57745/S8V5MM>, 2025.
- [13] Dimitris C. Paraskevopoulos, Gilbert Laporte, Panagiotis P. Repoussis, and Christos D. Tarantilis. Resource constrained routing and scheduling : Review and research prospects. *European Journal of Operational Research*, 263(3) :737–754, December 2017.
- [14] Victor Pillac, Christelle Guéret, and Andrés L. Medaglia. A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7) :1525–1535, October 2012.
- [15] Thomas Pouré, Elise Vareilles, Stéphanie Roussel, and Gauthier Picard. Configuration of heterogeneous agent fleet : a preliminary generic model. In *ConfWS'24 : 26th International Workshop on Configuration*. CEUR Workshop Proceedings, 2024.
- [16] Charles Prud'homme and Jean-Guillaume Fages. Choco-solver : A java library for constraint programming. *Journal of Open Source Software*, 7(78) :4708, October 2022.
- [17] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4) :455–472, November 2006.
- [18] Hamid Reza Sayarshad and Reza Tavakkoli-Moghaddam. Solving a multi periodic stochastic model of the rail-car fleet sizing by two-stage optimization formulation. *Applied Mathematical Modelling*, 34(5) :1164–1174, 2010.
- [19] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2) :254–265, April 1987.
- [20] Pascal Van Hentenryck. *The OPL optimization programming language*. MIT Press, Cambridge, MA, USA, 1999.
- [21] Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1) :11–12, January 1962.
- [22] Şule Yıldırım and Barış Yıldız. Electric bus fleet composition and scheduling. *Transportation Research Part C : Emerging Technologies*, 129 :103197, 2021.

Approche heuristique pour l'équilibrage de lignes d'assemblage avec minimisation du pic de consommation énergétique

Thiago G. Araujo¹, Zhifei Zheng², Matthieu Py¹, Sami Cherif²,
Rui Sá Shibasaki², Laurent Deroussi¹, Nathalie Grangeon¹, Chu-Min Li²

¹ Université Clermont Auvergne, CNRS, Clermont Auvergne INP, Mines Saint-Étienne,
LIMOS, 63000 Clermont-Ferrand, France

² Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

thiago.giachetto_de_araujo@uca.fr

Résumé

Dans un monde où les enjeux environnementaux sont de plus en plus importants, l'amélioration de l'efficacité énergétique des systèmes de production est cruciale. Cet article s'intéresse aux problèmes d'équilibrage de lignes d'assemblage, dans lesquels une stratégie possible consiste à minimiser les pics de consommation énergétique générés par l'ordonnancement des tâches choisis, problème dénommé SALB3PM. Pour résoudre ce problème, nous comparons expérimentalement deux méthodes : une méta-heuristique à redémarrages multiples, basée sur une recherche locale exploitant trois types de mouvements ; et une approche de résolution basée sur la Satisfiabilité Maximum (MaxSAT) issue de travaux précédents. Les expérimentations sont menées sur plus d'une centaine d'instances de la littérature et montrent l'efficacité des deux approches pour la résolution de SALB3PM.

Mots-clés

Ordonnancement, SALBP, Énergie, Métaheuristiques, MaxSAT.

Abstract

In a world where environmental concerns are increasingly central, improving the energy efficiency of production systems is crucial. This paper focuses on assembly line balancing problems, where the objective is to minimize the energy consumption peaks generated by task scheduling on assembly lines. This problem is known as the Simple Assembly Line Balancing Problem with Power Peak Minimization (SALB3PM). To solve this problem, we experimentally compare two methods: a multi-start metaheuristic based on local search using three types of moves, and a resolution approach based on Maximum Satisfiability (MaxSAT) derived from previous work. Experiments are conducted on more than a hundred benchmark instances and demonstrate the effectiveness of both approaches for solving SALB3PM.

Keywords

Scheduling, SALBP, Energy, Metaheuristics, MaxSAT.

1 Introduction

Les lignes d'assemblage constituent un système de production important. Elles se caractérisent par un agencement linéaire de postes de travail. Dans ce système, les produits se déplacent séquentiellement d'un poste de travail à l'autre, des tâches spécifiques étant exécutées à chaque poste jusqu'à l'assemblage complet du produit. Ces systèmes peuvent être modélisés sous forme de problèmes d'équilibrage de lignes d'assemblage simples (Simple Assembly Line Balancing Problems, SALBP) [2]. Les variantes SALBP les plus connues ont largement été étudiées dans la littérature [17, 16, 6]. Alors que les conséquences du réchauffement climatique affectent la population mondiale, la réduction des émissions de CO_2 devient impérative. Le rapport de 2024 de l'Agence Internationale de l'Énergie (AIE) a montré que le secteur industriel est responsable de 45% des émissions de CO_2 et de 39% de la consommation totale d'énergie dans le monde [1]. Il est donc essentiel d'améliorer l'efficacité énergétique des systèmes de production. Pour répondre à ces enjeux, de nouvelles variantes de SALBP intégrant des contraintes énergétiques ont récemment été proposées.

Un exemple est le problème SALBP avec minimisation du pic de consommation énergétique (ou simplement SALB3PM), introduit dans [8]. Dans ce problème, chaque tâche est associée à une consommation d'énergie constante, indépendante du poste de travail. L'objectif est de minimiser le pic de consommation énergétique, c'est-à-dire le maximum de consommation d'énergie sur l'ensemble des unités de temps. Des méthodes exactes ont été proposées pour résoudre SALB3PM dans [8, 14, 18]. Dans l'article original [8], une formulation en Programmation Linéaire en Nombres Entiers (PLNE) a été proposée. Dans [14], un algorithme exact basé sur des appels itératifs à des solveurs de satisfiabilité (SAT) a été introduit pour SALB3PM. Plus récemment, un modèle basé sur la satisfiabilité maximum (MaxSAT) a été proposé dans [18], et deux représentations booléennes différentes ont été testées. Les résultats expérimentaux

montrent que les formulations SAT et MaxSAT sont efficaces pour SALB3PM, surpassant les formulations PLNE existantes. Cependant, à mesure que la taille des instances augmente, ces approches exactes rencontrent des difficultés à résoudre les instances de manière optimale.

Étant donné la difficulté d'obtenir des solutions optimales, des approches métaheuristiques ont été proposées pour une variante de SALB3PM qui ne prend pas en compte les délais entre les tâches, à savoir le SALB3PM-ESD. Pour traiter ce problème, une métaheuristique basée sur une recherche locale évolutive à démarrages multiples (Multi-start Evolutionary Local Search, MS-ELS) a été mise en œuvre dans [11]. En étudiant la même variante que [11], un décodeur basé sur une formulation PLNE pour des heuristiques à base de permutations a été introduit dans [5]. Pour tester leur proposition, les auteurs ont implémenté un algorithme de recuit simulé simple avec le décodeur proposé. Cependant, cette heuristique ne peut pas être utilisée directement pour résoudre le problème SALB3PM classique.

Pour résoudre SALB3PM, on présente dans cet article deux méthodes heuristiques. La première, issue de travaux récents [18], consiste à modéliser le problème SALB3PM sous forme MaxSAT, et à lancer une méthode de résolution incomplète dédié au problème MaxSAT. La seconde, brièvement proposée dans [7], utilise une métaheuristique à redémarrages multiples avec trois différents types de voisinage. Les solutions initiales sont générées à l'aide d'une heuristique constructive basée sur les priorités, avec une fonction de priorité semi-aléatoire. Pour la recherche locale, nous adaptons deux structures de voisinage couramment utilisées dans la littérature, l'insertion de tâche et l'échange de tâches, et nous introduisons un troisième mouvement qui consiste à augmenter le délai des tâches.

L'article est organisé comme suit. La section 2 présente le problème SALB3PM. La section 3 présente la formulation MaxSAT du problème proposée dans [18]. La section 4 présente la métaheuristique proposée. La section 5 propose une étude expérimentale des méthodes présentées. Enfin, on conclut et on discute les travaux futurs dans la section 6.

2 Problème SALB3PM

Une ligne d'assemblage se compose d'un ensemble $\mathcal{M} = \{1, \dots, m\}$ de postes de travail disposés de manière linéaire. Pour fabriquer un produit, un ensemble $\mathcal{O} = \{1, \dots, n\}$ d'opérations indivisibles, appelées tâches, doit être exécuté. En raison de contraintes intrinsèques, certaines tâches ne peuvent être exécutées qu'après l'achèvement d'autres tâches. Par conséquent, les tâches doivent respecter des contraintes de précédence. Ces contraintes peuvent être représentées par un graphe de précédence, où chaque tâche est représentée par un nœud, et où un arc (i, j) existe si la tâche i doit être terminée avant

que la tâche j ne commence. Cette relation de précédence entre les tâches i et j est notée $i \prec j$.

Exemple 1 La figure 1 montre un graphe de précédence qui concerne 7 tâches. En particulier, pour que la tâche 6 puisse être exécutée, il faut que ses prédécesseurs directs (tâche 5) et indirects (tâches 1, 2 et 4) soient terminés. De manière similaire, la tâche 6 devra être terminée avant que ses successeurs directs (tâche 7) et ses successeurs indirects ne puissent commencer.

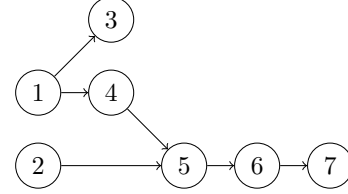


Figure 1: Exemple de graphe de précédence

Dans le problème d'équilibrage de lignes d'assemblage simple (Simple Assembly Line Balancing Problem, SALBP), chaque tâche $j \in \mathcal{O}$ possède un temps de traitement connu t_j , indépendant du poste auquel elle est affectée. Chaque tâche doit être affectée à un et un seul poste de travail de manière à ce que toutes les contraintes de précédence doivent être respectées : si $i \prec j$, alors soit les tâches i et j sont affectées au même poste, soit i est affectée à un poste situé avant celui de la tâche j sur la ligne. Ainsi, l'ensemble des tâches doit être réparti en ensembles disjoints S_k , où $k \in \mathcal{M}$. Étant donné une tâche $i \in S_a$ et une tâche $j \in S_b$, la contrainte de précédence $i \prec j$ impose que $a \leq b$. Le temps de traitement total du poste k , noté $t(S_k)$, correspond à la somme des temps de traitement des tâches qui lui sont affectées. Si un temps de cycle fixe c est défini, une solution est considérée comme faisable si $t(S_k) \leq c, \forall k \in \mathcal{M}$.

Selon la fonction objectif choisie, différentes variantes du problème SALBP peuvent être définies :

- Dans le problème SALBP-1, le temps de cycle c est donné, et l'objectif est de minimiser le nombre de postes de travail m .
- Dans le problème SALBP-2, le nombre de machines m est donné, et l'objectif est de minimiser le temps de cycle c .
- Dans le problème SALBP-F, le nombre de machines m et le temps de cycle c sont tous les deux donnés, et l'objectif est de déterminer s'il existe une solution faisable pour ces valeurs.
- Dans le problème SALBP-E, l'objectif est de maximiser l'efficacité, ce qui revient à minimiser le produit du nombre de machines m et du temps de cycle c [16].

Dans cet article, on s'intéresse au problème SALBP avec minimisation du pic de consommation d'énergie (SALB3PM). Cette nouvelle variante du problème SALBP a pour objectif de répondre davantage aux problématiques énergétiques de notre époque. Dans le problème SALB3PM, le nombre de machines m et le temps de cycle c sont donnés. De plus, chaque tâche j possède maintenant une consommation d'énergie constante par unité de temps w_j , qui est indépendante de sa station d'affectation. Il est donc désormais possible de calculer, pour chaque unité de temps t , la consommation énergétique totale sur cette unité de temps, qui est la somme des consommations énergétiques des tâches en cours d'exécution à cet instant. L'objectif du problème SALB3PM est d'ordonnancer chacune des tâches afin de minimiser le pic de consommation énergétique.

Exemple 2 On considère une instance du problème SALB3PM avec $m = 3$ postes de travail et un temps de cycle de $c = 6$ unités de temps. Le tableau 1 indique, pour chaque tâche, sa durée d'exécution et sa consommation énergétique par unité de temps. Le graphe de précedence est celui précédemment décrit dans la figure 1.

Tâche	1	2	3	4	5	6	7
Durée d'exécution	2	3	2	2	3	2	2
Consommation d'énergie	10	20	10	20	20	20	10

Table 1: Durée d'exécution et consommation énergétique de chaque tâche

Un exemple de solution qui respecte toutes les contraintes du problème SALB3PM est représenté dans la figure 2, où chaque rectangle représente une tâche dont la hauteur est sa consommation énergétique et sa longueur est sa durée d'exécution. Comme indiqué dans la figure 2 :

- Au temps 0, les tâches 1, 3 et 4 commencent, ce qui implique une consommation d'énergie de 40 par unité de temps.
- Au temps 2, les tâches 1, 3 et 4 se terminent, alors que les tâches 2, 5 et 6 commencent. Cela augmente la consommation d'énergie à 60 par unité de temps.
- Au temps 4, la tâche 6 se termine alors que la tâche 7 commence. Cela diminue la consommation d'énergie à 50 par unité de temps.
- Au temps 5, les tâches 2 et 5 terminent. Cela diminue fortement la consommation d'énergie à 10 par unité de temps.

Un exemple de solution optimale pour cette instance est proposé dans la figure 4, où l'on peut constater que les trois tâches avec la plus grosse consommation d'énergie (les tâches 2, 5 et 7) ne sont cette fois plus exécutées en même temps. On peut aussi noter que la consommation énergétique y est plus lissée, c'est à dire que la différence entre le pic de consommation énergétique

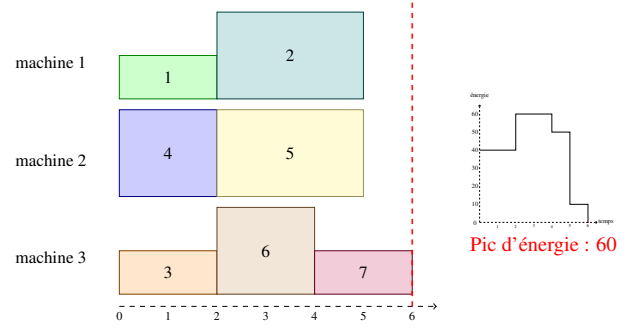


Figure 2: Solution réalisable pour le problème SALB3PM

(le moment où la consommation est la plus grande) et le creux de consommation énergétique (le moment où la consommation est la plus faible) est plus petite que dans la solution précédemment proposée. En outre, introduire des temps d'inactivité pour les machines semble être parfois nécessaire pour minimiser le pic de consommation, ce qui implique l'obtention d'ordonnancements non semi-actifs.

3 Résolution de SALB3PM grâce à une formulation MaxSAT

Dans cette section, on présente la formulation MaxSAT du problème SALB3PM initialement proposée dans [18]. Parmi les deux modèles proposés dans l'article original, on utilisera ici uniquement le deuxième modèle, basé sur une représentation binaire et dont les résultats expérimentaux donnent les meilleures performances. Pour plus de détails, le lecteur peut se référer à l'article [18]. On rappelle que le problème de satisfiabilité maximum (MaxSAT) est l'extension naturelle de la satisfiabilité (SAT) en problème d'optimisation [12]. Dans la suite, on utilisera plus particulièrement la variante partielle pondérée de MaxSAT qui prend en entrée une formule bipartite pondérée $\phi = D \cup S$ contenant des variables booléennes, où D est l'ensemble des clauses dures qui doivent être satisfaites, comme dans SAT et S est l'ensemble des clauses souples pondérées (c, w_c) avec w_c un poids entier positif associé à la clause c . L'objectif de MaxSAT partiel pondéré est de trouver une affectation des variables de la formule qui maximise la somme des poids des clauses souples satisfaites tout en satisfaisant toutes les clauses dures. Avant d'introduire le modèle MaxSAT, on mentionne également l'existence de contraintes pseudo-bouliennes ayant la forme $\sum_j a_j \cdot l_j \circ b$ où $a_j \in \mathbb{N}$, $b \in \mathbb{N}$, l_j est un littéral et $\circ \in \{=, \geq, \leq\}$ et pouvant être efficacement encodées sous forme clausale [15].

Pour introduire le modèle MaxSAT, le temps est discrétisé en l'ensemble des unités de temps $T = \{0, 1, \dots, c-1\}$. On note $T^i = \{0, \dots, c-t_i\}$ l'ensemble des unités de temps où il est possible de faire débuter la tâche $i \in O$. La valeur du pic de consommation énergétique appartient forcément à l'intervalle $[LB, UB]$, où LB est une borne inférieure calculée à partir de la plus haute

consommation énergétique parmi toutes les tâches et UB est une borne supérieure calculée à partir des m plus hautes consommations énergétiques. Plus formellement, on $LB = \max_{i \in \mathcal{O}} w_i$ et $UB = \sum_{w_i \in W_m} w_i$.

La formulation MaxSAT utilise quatre ensemble de variables de décision booléennes :

- **Les variables d'affectations** $X_{i,s}$ qui, pour une tâche $i \in \mathcal{O}$ et une machine $s \in \mathcal{M}$, sont affectées à *Vrai* si la tâche i est assignée à la machine s .
- **Les variables d'ordonnancement** $B_{i,t}$ qui, pour une tâche $i \in \mathcal{O}$ et pour une unité de temps $t \in T$, sont affectées à *Vrai* si la tâche i commence à l'unité de temps t .
- **Les variables d'activité** $A_{i,t}$ qui, pour une tâche $i \in \mathcal{O}$ et une unité de temps $t \in T$, sont affectées à *Vrai* si la tâche i est en cours d'exécution à l'unité de temps t .
- **Les variables d'énergie** $\text{bin}U_j$ qui encodent la valeur du pic de consommation énergétique. Chaque variable $\text{bin}U_j$, pour une valeur $j \in \{0, \dots, \lceil \log_2 UB \rceil\}$, représente la valeur du $j^{\text{ème}}$ bit du pic d'énergie de la solution. Le pic de consommation énergétique W_{peak} est calculé de la manière suivante :

$$W_{\text{peak}} = \sum_{j \in [0, \lceil \log_2 UB \rceil]} 2^j \cdot \text{bin}U_j \quad (1)$$

Le modèle MaxSAT est représenté en figure 3 avec les clauses souples permettant de minimiser le pic de consommation énergétique les clauses dures garantissant le respect des contraintes intrinsèques du problème :

- Les clauses souples pondérées (ligne 2) comptabilisent lors du pic d'énergie la quantité d'énergie économisée par rapport à la borne supérieure UB (et l'objectif est de maximiser cette quantité, autrement dit de minimiser le pic de consommation énergétique).
- Les clauses dures permettent de :
 - garantir que chaque tâche soit affectée à exactement une machine (contrainte 3).
 - garantir le respect des contraintes de précédence (contraintes 4 et 5).
 - garantir que chaque tâche débute une et une seule fois et seulement dans l'intervalle de temps qui lui permet de terminer avant la fin du temps de cycle (contraintes 6 et 7).
 - lier les variables d'ordonnancement et les variables d'activité (contrainte 8).
 - garantir le non-chevauchement des tâches (contrainte 9).
 - gérer le calcul du pic de consommation énergétique (contraintes 10 et 11), avec $UB' = \sum_{j=0}^{\lceil \log_2 UB \rceil} 2^j$.

Clauses souples:

$$(\overline{\text{bin}U_j}, 2^j) \quad \forall j \in [0, \lceil \log_2 UB \rceil] \quad (2)$$

Clauses dures:

$$\sum_{s \in \mathcal{M}} X_{i,s} = 1 \quad \forall i \in \mathcal{O} \quad (3)$$

$$\overline{X_{i,s1}} \vee \overline{X_{j,s2}} \quad \forall (i, j, s1, s2) \in \mathcal{O}^2 \times \mathcal{M}^2 \text{ s.t } i \prec j \text{ and } s1 > s2 \quad (4)$$

$$\overline{X_{i,s}} \vee \overline{X_{j,s}} \vee \overline{B_{i,t1}} \vee \overline{B_{j,t2}} \quad \forall (i, j, s, t1, t2) \in \mathcal{O}^2 \times \mathcal{M} \times T^2 \text{ s.t } i \prec j \text{ and } t1 > t2 \quad (5)$$

$$\sum_{t \in T} B_{i,t} = 1 \quad \forall i \in \mathcal{O} \quad (6)$$

$$\overline{B_{i,t}} \quad \forall (i, t) \in \mathcal{O} \times T \setminus T^i \quad (7)$$

$$\overline{B_{i,t}} \vee A_{i,t+\epsilon} \quad \forall (i, t, \epsilon) \in \mathcal{O} \times T^i \times \{0, \dots, t_i - 1\} \quad (8)$$

$$\overline{X_{i,s}} \vee \overline{X_{j,s}} \vee \overline{A_{i,t}} \vee \overline{A_{j,t}} \quad \forall (i, j, s, t) \in \mathcal{O}^2 \times \mathcal{M} \times T \text{ s.t } i < j \quad (9)$$

$$\sum_{j \in [0, \lceil \log_2 UB \rceil]} 2^j \text{bin}U_j \geq LB \quad \forall t \in T \quad (10)$$

$$\sum_{i \in \mathcal{O}} W_i * A_{i,t} + \sum_{j \in [0, \lceil \log_2 UB \rceil]} 2^j * \text{bin}U_j \leq UB' \quad \forall t \in T \quad (11)$$

Figure 3: Modélisation MaxSAT du problème SALB3PM

4 Résolution métaheuristique de SALBP3PM

Trouver des solutions optimales pour le problème SALB3PM est difficile [8, 14] ; c'est pourquoi nous avons étudié une approche métaheuristique. La méthode proposée repose sur une méthode à redémarrages multiples qui combine une génération semi-aléatoire de solutions et une recherche locale utilisant trois structures de voisinage différentes. L'approche proposée est présentée comme suit. La section 4.1 introduit l'approche proposée à démarrages multiples. Les sections 4.2 et 4.3 décrivent respectivement comment une solution est représentée et évaluée. La section 4.4 présente l'heuristique constructive semi-aléatoire utilisée pour générer les solutions initiales. Dans la section 4.5, la structure de voisinage est définie. Enfin, la section 4.6 décrit comment la recherche locale est effectuée sur les solutions générées.

4.1 Présentation de la métaheuristique

À l'origine, les procédures à redémarrages multiples étaient utilisées pour exploiter une procédure de recherche locale. Plusieurs solutions initiales sont générées aléatoirement, puis la recherche locale est appliquée [13]. L'algorithme 1 présente les différentes étapes d'une telle procédure pour un problème de minimisation. La procédure prend en entrée une fonction d'évaluation $f(\cdot)$ ainsi qu'un nombre maximal d'itérations. Tout d'abord, l'algorithme considère qu'il n'a aucune meilleure solution connue avec un pic de consommation énergétique arbitrairement infini (ligne 2). Itérativement, une valeur aléatoire est tirée à partir d'une distribution uniforme. Cette valeur est utilisée dans la fonction d'évaluation (ligne 4) et son rôle sera décrit dans la section 4.3. Une solution aléatoire est générée,

Algorithm 1 Approche à redémarrages multiples

```

1: function MÉTAHEURISTIQUE( $f(\cdot)$ ,  $nb\_runs$ )
2:    $S \leftarrow \emptyset$ ,  $f(S) \leftarrow \infty$ 
3:   while on a fait moins de  $nb\_runs$  itérations do
4:      $W \leftarrow U(0, \max_{j \in \mathcal{O}}(w_j))$ 
5:      $S_{courante} \leftarrow Generer\_solution()$ 
6:      $S_{courante} \leftarrow Rech\_Locale(S_{courante})$ 
7:     if  $f(S_{courante}) < f(S)$  then
8:        $S \leftarrow S_{courante}$ 
9:     end if
10:  end while
11:  return  $S$ 
12: end function

```

puis une procédure de recherche locale est appliquée à cette solution (lignes 5 et 6). Si la solution courante trouvée par la recherche locale est meilleure que la meilleure solution trouvée, celle-ci est mise à jour (lignes 7 et 9). La boucle allant de la ligne 3 à la ligne 10 est exécutée autant de fois qu'indiqué par le paramètre ' nb_runs '. Enfin, la meilleure solution est retournée à la fin de l'algorithme.

4.2 Représentation d'une solution

Représenter une solution est un facteur clé pour appliquer avec succès une métaheuristique à un problème. Dans l'approche proposée, nous considérons une représentation indirecte d'une solution. Ce type de représentation a été utilisé dans [11], où un vecteur d'affectation et un vecteur de séquençement des opérations étaient employés. Nous étendons cette représentation en introduisant un troisième vecteur qui constitue notre contribution. Plus précisément, nous utilisons trois vecteurs pour représenter une solution (comme illustré dans la figure 4) :

- Le premier vecteur, V , contient l'ordre d'exécution des tâches.
- Le deuxième vecteur, A , est un vecteur d'affectation qui précise le poste de travail attribué à chaque tâche.
- Le troisième vecteur, D , contient le délai de chaque tâche. Le délai d'une tâche j est défini comme le temps d'inactivité pendant lequel la tâche j est différée après la fin de la tâche précédente sur le même poste de travail.

Le processus de décodage des trois vecteurs d'une solution utilise le vecteur A pour affecter toutes les tâches aux postes de travail. L'ordre des tâches au sein des postes de travail suit l'ordre défini dans V , et le délai de chaque tâche est donné par D . À l'issue de cette procédure de décodage, il est possible de déterminer les instants de début et de fin de chaque tâche.

4.3 Évaluation d'une solution

La fonction objectif du problème est de minimiser le pic de consommation énergétique. Étant donné que des solutions violant la contrainte de temps de cycle peuvent être générées par notre algorithme, on utilise une fonction

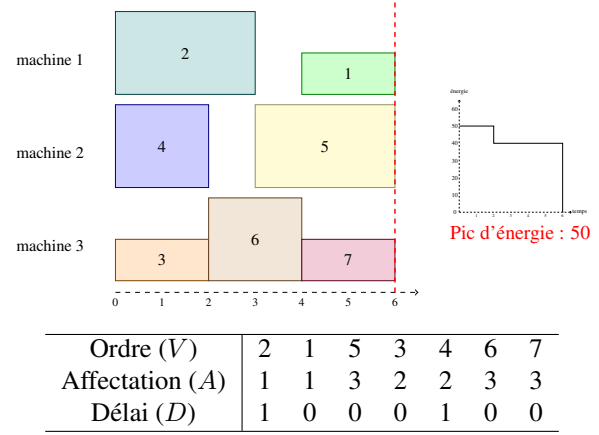


Figure 4: Exemple de solution optimale et de sa représentation

d'évaluation pénalisant les solutions non réalisables. La pénalité associée à une solution s est calculée comme le temps total par lequel le temps de cycle est dépassé. L'équation 12 résume le calcul de la pénalité, où d_j représente le délai de la tâche j .

$$penalty(s) = \sum_{k \in \mathcal{M}} \max(0, \sum_{j \in S_k} (t_j + d_j) - c) \quad (12)$$

L'équation 13 définit la fonction d'évaluation. Soit $power_peak(s)$ le pic de consommation énergétique de la solution s , $penalty(s)$ la pénalité calculée selon l'équation 12, et W un facteur de pénalité tiré d'une distribution uniforme $U(0, \max_{j \in \mathcal{O}}(w_j))$. La valeur de W est mise à jour à chaque itération, comme indiqué à la ligne 4.

$$f(s) = power_peak(s) + penalty(s) \cdot W \quad (13)$$

4.4 Génération d'une solution initiale

Les heuristiques constructives basées sur des règles de priorité pour SALBP ont été largement étudiées dans la littérature [17, 6]. Les méthodes basées sur des priorités utilisent un poids associé à chaque tâche pour l'ordonner. Comme l'algorithme à redémarrages multiples dépend de solutions initiales à la fois variées et de bonne qualité, nous utilisons une version semi-aléatoire de l'heuristique constructive basée sur le temps de traitement le plus long pour générer les solutions initiales [6].

Pour chaque tâche, la règle de priorité est calculée selon l'équation 14, où $x \in [0, 1]$ est une valeur aléatoire générée pour chaque tâche, et où le paramètre $\alpha \in [0, 1]$ contrôle le degré d'aléatoire de la fonction de priorité. Les durées de traitement minimale et maximale sont respectivement notées t_{min} et t_{max} . Pour $\alpha = 0$, la priorité est entièrement aléatoire, tandis que pour $\alpha = 1$, la priorité est purement gloutonne.

$$priority(j) = (1 - \alpha)x + \alpha \frac{t_j - t_{min}}{t_{max} - t_{min}} \quad (14)$$

La solution générée respecte les contraintes de précédence, le délai est nul pour toutes les tâches, et tous les postes de travail, à l'exception du dernier, respectent la contrainte de temps de cycle. Pour comprendre la procédure d'affectation, certaines définitions sont nécessaires. Une tâche est dite disponible si toutes ses tâches prédécesseuses directes ont été affectées. Une tâche j peut être affectée au poste de travail k si :

- Tous ses prédécesseurs ont été affectés aux postes $1, \dots, k$, et
- Le temps total de traitement du poste k , additionné à la durée de la tâche j , ne dépasse pas le temps de cycle.

Toutefois, pour le dernier poste de travail, la contrainte de temps de cycle est relâchée. La procédure d'affectation est représentée dans l'algorithme 2 [6, 9].

À la ligne 2, la liste des tâches à affecter ($\mathcal{T}$), le compteur de postes de travail (k) et le temps disponible (CT) sont initialisés. La boucle des lignes 3 à 16 se poursuit tant que $\mathcal{T}$ n'est pas vide. À la ligne 4, la liste des tâches disponibles, LCT , est mise à jour avec toutes les tâches pouvant être affectées au poste k . Si LCT n'est pas vide, aux lignes 6 à 9, la tâche ayant la priorité la plus élevée, notée $next$, est sélectionnée dans LCT . Cette tâche est ensuite retirée de $\mathcal{T}$ et affectée au poste k . Le temps disponible CT est décrémenté de w_{next} . Si LCT est vide, le temps disponible CT est réinitialisé à la ligne 11. Si le nombre de postes ouverts est inférieur à m (c'est-à-dire $k < m$), le compteur de postes k est incrémenté à la ligne 13. Pour le dernier poste de travail ($k = m$), les tâches peuvent y être affectées sans respecter la contrainte de temps de cycle.

Algorithm 2 Procédure d'affectation

```

1: function GÉNÉRER_SOLUTION( $f(\cdot)$ ,  $nb\_runs$ )
2:    $\mathcal{T} \leftarrow \{1, \dots, n\}$ ,  $k \leftarrow 1$ ,  $CT \leftarrow c$ 
3:   while  $\mathcal{T} \neq \emptyset$  do
4:      $LCT = \{j \in \mathcal{T}, t_j \leq CT, j \text{ est disponible}\}$ 
5:     if  $LCT \neq \emptyset$  then
6:        $next \leftarrow$  tâche prioritaire dans  $LCT$ 
7:        $LCT \leftarrow LCT \setminus \{next\}$ 
8:       Affecter  $next$  au poste de travail  $k$ 
9:        $CT \leftarrow CT - w_{next}$ 
10:    else
11:       $CT \leftarrow c$ 
12:      if  $k < m$  then
13:         $k \leftarrow k + 1$  ▷ Ouverture de poste
14:      end if
15:    end if
16:  end while
17: end function

```

4.5 Structure de voisinage

Trois structures de voisinage sont considérées dans l'approche actuelle. La première est basée sur l'insertion

de tâches ($\mathcal{N}_{insertion}$), la deuxième sur l'échange de tâches ($\mathcal{N}_{swap}$), et la troisième sur l'augmentation du délai d'une tâche ($\mathcal{N}_{delay}$). Les deux premiers mouvements peuvent violer les contraintes de temps de cycle.

4.5.1 Insertion de tâche

Étant donnée une solution S et une tâche j :

- la position du prédécesseur le plus proche de la tâche j dans $\mathcal{O}$ est notée $pred_{near}(j)$, et son poste de travail est $ws_{pred}(j)$. Si la tâche j n'a pas de prédécesseurs, alors $pred_{near}(j) = 0$ et $ws_{pred}(j) = 1$;
- la position du successeur le plus proche de la tâche j dans $\mathcal{O}$ est notée $suc_{near}(j)$, et son poste de travail est $ws_{suc}(j)$. Si la tâche j n'a pas de successeurs, alors $suc_{near}(j) = n$ et $ws_{suc}(j) = m$.

Une solution voisine est obtenue en retirant la tâche j de sa position actuelle, puis en l'insérant dans l'intervalle $[pred_{near}(j) + 1, suc_{near}(j) - 1]$, tout en l'affectant à un poste de travail dans l'intervalle $[ws_{pred}(j), ws_{suc}(j)]$. Le retard de la tâche j est alors réinitialisé à zéro. Ce voisinage est de taille $O(mn^2)$.

4.5.2 Échange de deux tâches

Étant donnée une solution S et une tâche j_1 , on calcule $pred_{near}(j_1)$ et $suc_{near}(j_1)$ comme décrit dans la Section 4.5.1. On sélectionne une tâche $j_2 \in [pred_{near}(j_1) + 1, suc_{near}(j_1) - 1]$, telle que $j_1 \in [pred_{near}(j_2) + 1, suc_{near}(j_2) - 1]$. Une solution voisine est obtenue en échangeant les positions de j_1 et j_2 dans le vecteur V ainsi que leurs affectations aux postes de travail. Les retards des tâches j_1 et j_2 sont alors réinitialisés à zéro. Ce voisinage est de taille $O(n^2)$.

4.5.3 Augmenter le délai d'une tâche

Étant donnée une solution faisable S et un poste de travail k dont le temps total de traitement est inférieur au temps de cycle c , on sélectionne une tâche j dans le poste de travail k . Une solution voisine est obtenue en incrémentant le délai de la tâche j . L'incrément ne doit pas faire terminer la dernière tâche du poste de travail k après le temps de cycle. Ce voisinage est de taille $O(nc)$.

4.6 Recherche Locale

Une procédure de recherche locale est utilisée pour améliorer les solutions générées. Les trois voisinages décrits ci-dessus sont employés dans la recherche locale. Une stratégie de première amélioration est adoptée (nommée *first_improvement*). Pour chaque voisinage, étant donnée une solution, ses voisins sont évalués dans un ordre aléatoire jusqu'à ce qu'une solution améliorante soit trouvée. Cela signifie que si la solution initiale est un optimum local pour un voisinage donné, tous ses voisins seront évalués. La recherche locale s'arrête lorsqu'un optimum local est trouvé pour les trois voisinages.

Cette procédure est représentée dans l'algorithme 3. L'algorithme fonctionne comme suit. Une solution S et

Algorithm 3 Algorithme de recherche locale

```

1: function RECHERCHE_LOCALE( $S, f(\cdot)$ )
2:    $eval\_begin \leftarrow f(S), is\_local\_optimum \leftarrow false$ 
3:   while  $!is\_local\_optimum$  do
4:     if  $x > 0.5 \mid x \sim \mathcal{U}(0, 1)$  then
5:        $S \leftarrow first\_improvement(S, \mathcal{N}_{swap})$ 
6:        $S \leftarrow first\_improvement(S, \mathcal{N}_{insertion})$ 
7:     else
8:        $S \leftarrow first\_improvement(S, \mathcal{N}_{insertion})$ 
9:        $S \leftarrow first\_improvement(S, \mathcal{N}_{swap})$ 
10:    end if
11:    if  $S$  est une solution faisable then
12:       $S \leftarrow first\_improvement(S, \mathcal{N}_{delay})$ 
13:    end if
14:    if  $f(S) = eval\_begin$  then
15:       $is\_local\_optimum \leftarrow 1$ 
16:    else
17:       $eval\_begin \leftarrow f(S)$ 
18:    end if
19:  end while
20:  return  $S$ 
21: end function

```

une fonction d'évaluation $f(\cdot)$ sont fournies en entrée. La boucle des lignes 3 à 19 continue jusqu'à ce qu'aucune solution voisine améliorante ne soit trouvée. À l'intérieur de cette boucle, la ligne 4 sélectionne aléatoirement l'ordre dans lequel $\mathcal{N}_{insertion}$ ou $\mathcal{N}_{swap}$ est appliqué. Quelque soit cet ordre, la fonction $first_improvement$ retourne le premier voisin de S dont la valeur de la solution est meilleur que S . Ensuite, si S est une solution réalisable, le voisinage $\mathcal{N}_{delay}$ est utilisé à la ligne 12 pour tenter d'en améliorer la valeur. Les lignes 14 à 18 vérifient si aucun des trois voisinages n'a permis de trouver une meilleure solution, auquel cas, étant coincé dans un optimum local, la recherche locale est arrêtée et la meilleure solution est renvoyée.

5 Expérimentations

Un ensemble d'instances de référence¹, étendu à partir de travaux précédents, a été utilisé dans nos expérimentations. Il contient 130 instances, réparties en 13 familles de 10 instances chacune. Pour chaque famille, 5 instances possèdent un nombre fixe de postes de travail m , et le temps de cycle c est calculé à l'aide d'un solveur SALBP-2 (ces instances sont appelées *normales*) ; les 5 autres instances (appelées *étendues*) ont été générées avec $c' = \lceil 1,3 \cdot c \rceil$. La consommation énergétique w_j de chaque tâche a été générée à partir d'une distribution uniforme $U(5, 50)$ [8, 14, 18]. Le tableau 2 résume les caractéristiques des instances. Les deux premières colonnes donnent le nom de la famille d'instances et le nombre de tâches. Le nombre minimum et maximum de machines, ainsi que le temps de cycle, sont également fournis, séparés par un tiret. La métaheuristique proposée a été implémentée en C++ et compilée avec GNU C++ 13.2.0 en utilisant les options -O3

¹<https://github.com/ZZFreya/SALB3PM/tree/main/benchmarks/1993>

et -march=native. La méthode MaxSAT, présentée dans la section 3 et issue de [18], a été implémentée en Python et utilise le package PySAT [10]. La méthode est lancée avec un solveur MaxSAT exact (afin de voir les instances pour lesquelles la valeur de la solution optimale est trouvée), ainsi qu'un solveur heuristique :

- **MaxHS** [4] (solveur MaxSAT exact) : Ce solveur repose sur une architecture hybride combinant les techniques modernes de résolution SAT avec des méthodes d'optimisation en programmation entière.
- **NuWLS-c** [3] (solveur MaxSAT heuristique) : Ce solveur utilise une approche par portfolio combinant deux moteurs : une recherche locale MaxSAT avancée et le MaxSAT TT-Open-WBO-Inc. La version que nous avons utilisée a remporté la première place dans les quatre catégories incomplètes de l'évaluation MaxSAT 2023².

L'ensemble des expériences a été réalisé sur la plateforme MatriCS³, sur un serveur fonctionnant sous Rocky Linux 8.6, équipé de processeurs Intel Xeon E5-2680 v4 cadencés jusqu'à 2,40 GHz. Un seul thread a été utilisé pour toutes les expériences. Pour chaque instance, la métaheuristique proposée a été exécutée 10 fois. La valeur α du générateur de solutions initiales a été fixée à 0,3. Pour la méthode exacte, la limite de temps a été fixée à 7200 secondes, et pour les méthodes heuristiques (MaxSAT avec NuWLS-c et la métaheuristique introduite), elle a été fixée à 1000 secondes.

Le Tableau 3 résume les résultats sur deux catégories d'instances : les instances dont les solutions optimales sont connues (*OPT*) totalisant 83 instances, et l'ensemble complet des 130 instances (*ALL*). Pour l'ensemble complet, la métaheuristique est comparée avec la valeur de la solution optimale. Pour les instances dont les solutions optimales sont connues, les résultats de la méta-heuristique (*MetaH*) et de la résolution MaxSAT heuristique sont comparés, avec les métriques suivantes :

- Les écarts relatifs maximum (gap.max) et moyen (gap.avg) par rapport à la meilleure solution connue pour chaque instance;
- Le pourcentage d'instances pour lesquelles chaque méthode a obtenu :
 - la meilleure solution (#best)
 - la meilleure solution de façon consistante sur tous les lancements (#best10)
 - au moins une solution faisable (#feasible)
 - des solutions faisables à chaque lancement (#feasible10)

²<https://maxsat-evaluations.github.io/>

³<https://www.matrics.u-picardie.fr>

Table 2: Résumé des résultats des heuristiques sur chaque famille d'instances

name	#tasks	#machines	normal					extended				
			cycle	$\overline{gap}$ (%)	$\bar{t}$ (s)	#iterations	feas. (%)	cycle	$\overline{gap}$ (%)	$\bar{t}$ (s)	#iterations	feas. (%)
MERTENS	7	2 - 6	6 - 15	0.00	0.0	1.8	100.0	8 - 20	0.00	0.3	1122.8	100.0
BOWMAN8	8	3 - 7	17 - 28	0.00	0.0	2.8	100.0	23 - 37	3.87	0.1	271.5	100.0
JAESCHKE	9	4 - 8	6 - 10	0.00	0.0	5.1	100.0	8 - 13	5.63	4.1	16119.5	100.0
JACKSON	11	2 - 6	9 - 23	0.00	89.0	382136.4	100.0	12 - 30	1.38	8.6	19664.6	100.0
MANSOOR	11	1 - 5	45 - 185	0.00	0.1	103.0	100.0	59 - 241	9.75	24.0	40786.4	100.0
MITCHELL	21	3 - 7	16 - 35	0.36	35.0	31970.8	100.0	21 - 46	3.90	56.2	33399.2	100.0
ROSZIEG	25	5 - 9	16 - 26	1.71	139.5	96666.8	100.0	21 - 34	-7.57	270.7	112665.2	100.0
HESKIA	28	2 - 6	171 - 512	-1.65	385.6	48887.4	100.0	223 - 666	-31.68	303.3	8962.0	100.0
BUXEY	29	8 - 12	28 - 41	2.03	315.7	255088.0	100.0	37 - 54	-19.28	395.2	41585.9	100.0
SAWYER30	30	8 - 12	28 - 41	3.74	465.6	348708.0	100.0	37 - 54	-17.86	365.5	30136.5	100.0
GUNTHER	35	8 - 12	44 - 63	2.31	392.7	152396.7	100.0	58 - 82	-31.86	399.4	34261.2	100.0
WARNECKE	58	17 - 21	76 - 92	-	-	-	0.0	99 - 120	-35.99	428.4	3458.1	100.0
LUTZ2	89	22 - 26	19 - 23	-14.13	471.3	61436.6	80.0	25 - 30	-37.60	440.6	6246.6	100.0
				-0.21	184.8	117344.3	90.8		-12.10	207.4	26821.5	100.0

Métrique (%)	ALL	OPT	
	MetaH	MetaH	MaxSAT
gap.max	24.07	24.07	107.32
gap.avg	-6.50	2.31	4.27
#best	63.08	51.81	80.72
#best10	62.31	50.60	-
#feasible	95.38	100.00	100.00
#feasible10	95.38	100.00	-

Table 3: Résumé des résultats des heuristiques sur les 130 instances

Le Tableau 2 détaille les résultats de la métaheuristique sur chaque famille d'instances. Pour les instances normales et étendues, nous fournissons l'écart moyen entre la solution trouvée par la métaheuristique et la meilleure solution connue, le temps moyen ($\bar{t}$) mis par la métaheuristique pour trouver sa meilleure solution et le nombre moyen d'itérations ($\#iterations$) nécessaire à la métaheuristique pour trouver sa meilleure solution. Nous indiquons également le pourcentage de solutions réalisables trouvées par la métaheuristique.

En ce qui concerne les instances dont les solution optimales sont connues, nous fournissons des comparaisons détaillées entre notre méthode proposée et la formulation MaxSAT résolue via une approche heuristique, avec des métriques calculées par rapport aux valeurs des solutions optimales connues. Pour les autres instances, les métriques sont calculées en comparant les performances de notre méthode aux meilleures solutions obtenues par la formulation MaxSAT, qu'elle soit résolue exactement ou heuristiquement.

Les résultats expérimentaux présentés dans le Tableau 3 montrent que la métaheuristique proposée a égalé ou amélioré les meilleures solutions connues lors des 10 exécutions indépendantes pour 81 instances (soit 62,31% des instances). Si l'on considère les cas où la meilleure solution connue a été atteinte ou dépassée dans au moins une exécution, ce nombre passe à 82 instances (soit 63,08% des instances). En ce qui concerne la

faisabilité, l'algorithme a trouvé des solutions réalisables dans toutes les exécutions pour 124 instances (95,38% des instances), et n'a pas produit de solutions réalisables pour 6 instances difficiles (4,62% des instances). L'analyse comparative montre que, pour les instances avec des solutions optimales connues, notre méthode surpasse le solveur heuristique MaxSAT en termes de gaps moyens et maximums d'optimalité. Toutefois, le solveur MaxSAT heuristique a identifié des solutions optimales pour 28,92% d'instances supplémentaires. Une analyse plus approfondie du Tableau 2 révèle que les performances de la métaheuristique s'améliorent considérablement pour les instances de plus grande taille, avec des gains moyens d'au moins 17% par rapport à la meilleure solution connue pour les cas comportant plus de 25 tâches et des temps de cycle étendus.

6 Conclusion

Dans cet article, nous avons proposé une approche métaheuristique pour le problème d'équilibrage de ligne d'assemblage avec minimisation des pics de consommation énergétique (SALB3PM). Nous nous sommes intéressés à deux méthodes : une métaheuristique à redémarrages multiples contenant une recherche locale exploitant trois structures de voisinage ; et une formulation MaxSAT issue de travaux précédents [18]. Les résultats expérimentaux montrent que la nouvelle métaheuristique est efficace pour trouver des solutions de haute qualité, en égalant ou en améliorant la meilleure solution connue pour 62,31% des instances lors de toutes les exécutions, et en obtenant des solutions réalisables pour 95,38% des instances. De plus, pour les instances comportant plus de 25 tâches et des temps de cycle étendus, la métaheuristique améliore la meilleure solution connue d'au moins 17%. Les approches MaxSAT ont quant à elles des résultats qui se confirment pertinents par comparaison avec la métaheuristique.

Une partie des travaux futurs concerne l'amélioration de la métaheuristique via l'ajout d'autres structures de voisinage afin d'examiner de nouveaux espaces de solutions, la prise en compte de l'historique des solutions pour accélérer le processus de recherche, et l'étude de représentations

alternatives des solutions. Nous prévoyons également de réguler automatiquement le paramètre α au cours de la recherche, afin d'équilibrer dynamiquement l'exploration et l'exploitation. Par ailleurs, pour les instances difficiles où il est compliqué de trouver des solutions faisables, une piste consiste à relâcher la contrainte sur le nombre de postes de travail afin de faciliter la recherche. Enfin, compte tenu de la pertinence à la fois de la métaheuristique et de l'approche MaxSAT et du comportement différent de chacun en fonction de la famille d'instances étudiée, nous comptons explorer l'hybridation entre les métaheuristicues et les méthodes exactes. Une direction possible serait de résoudre le sous-problème de détermination des délais des tâches à l'aide d'une formulation MaxSAT, pour obtenir des solutions plus précises et plus efficaces.

Remerciements

Ce travail a été pour partie financé grâce à l'appui du Centre International de Recherche "Innovative Transportation and Production Systems". Ce travail a également bénéficié d'un accès aux ressources de calcul de haute performance (HPC) de la plateforme MatriCS de l'Université de Picardie Jules Verne, cofinancée par l'Union européenne via le Fonds Européen de Développement Régional (FEDER) et par le Conseil régional des Hauts-de-France.

References

- [1] IEA (2024). World Energy Outlook 2024. Technical Report Annex A, International Energy Association, Paris, 2024.
- [2] İlker Baybars. A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem. *Management Science*, 32(8):909–932, August 1986.
- [3] Yi Chu, Shaowei Cai, and Chuan Luo. Nuwls-c-2023: Solver description. *MaxSAT Evaluation 2023*, page 23, 2023.
- [4] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by Solving a Sequence of Simpler SAT Instances. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *LNCS*, pages 225–239. Springer, 2011.
- [5] Xavier Delorme, Paolo Gianessi, and Damien Lamy. A new Decoder for Permutation-based Heuristics to Minimize Power Peak in the Assembly Line Balancing. *IFAC-PapersOnLine*, 56(2):3704–3709, 2023.
- [6] Masood Fathi, Dalila Benedita Machado Martins Fontes, Matias Urenda Moris, and Morteza Ghobakhloo. Assembly line balancing problem: A comparative evaluation of heuristics and a computational assessment of objectives. *Journal of Modelling in Management*, 13(2):455–474, May 2018.
- [7] Thiago Giachetto de Araujo, Matthieu Py, Laurent Deroussi, and Nathalie Grangeon. A metaheuristic for the Simple Assembly Line Balancing Problem with Power Peak Minimization. In *26ème édition du congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2025)*, 2024.
- [8] Paolo Gianessi, Xavier Delorme, and Oussama Masmoudi. Simple Assembly Line Balancing Problem with Power Peak Minimization. In Farhad Ameri, Kathryn E. Stecke, Gregor von Cieminski, and Dimitris Kiritsis, editors, *Advances in Production Management Systems. Production Management for the Factory of the Future*, volume 566, pages 239–247, Austin, TX, United States, September 2019. Springer International Publishing.
- [9] Michel Gourgand, Nathalie Grangeon, and Sylvie Norre. Metaheuristics based on Bin Packing for the line balancing problem. *RAIRO - Operations Research*, 41(2):193–211, April 2007.
- [10] Alexey Ignatiev, António Morgado, and João Marques-Silva. Pysat: A python toolkit for prototyping with SAT oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2018.
- [11] Damien Lamy, Xavier Delorme, and Paolo Gianessi. Line Balancing and Sequencing for Peak Power Minimization. *IFAC-PapersOnLine*, 53(2):10411–10416, 2020.
- [12] Chu Min Li and Felip Manyà. MaxSAT, Hard and Soft Constraints. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 903–927. IOS Press, 2021.
- [13] Rafael Martí, Ricardo Aceves, Maria Teresa León, Jose M. Moreno-Vega, and Abraham Duarte. Intelligent Multi-Start Methods. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, pages 221–243. Springer International Publishing, Cham, 2019.
- [14] Matthieu Py, Arnaud Tuyaba, Laurent Deroussi, Nathalie Grangeon, and Sylvie Norre. Application of SAT to the Simple Assembly Line Balancing Problem with Power Peak Minimization. In *15th Pragmatics of SAT international workshop, a workshop of the 27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*, 2024.
- [15] Olivier Roussel and Vasco Manquinho. Pseudo-boolean and cardinality constraints. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1087–1129. IOS Press, 2021.
- [16] Armin Scholl and Christian Becker. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168(3):666–693, February 2006.
- [17] Armin Scholl and Stefan Voss. Simple assembly line balancing - Heuristic approaches. *Journal of Heuristics*, 2(3):217–244, 1997.
- [18] Zhifei Zheng, Sami Cherif, and Rui Sá Shibasaki. Optimizing Power Peaks in Simple Assembly Line Balancing Through Maximum Satisfiability. In *2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 363–370, 2024.

Quel codage CNF pour les contraintes de concordance de motifs d'Eternity II ?

Olivier Bailleux¹

¹ Université Bourgogne Europe, LIB

olivier.bailleux@ube.fr

Résumé

Les puzzles de type Edge-matching — dont Eternity II (ir-résolu depuis 2007) est un exemple emblématique — sont l'archétype du problème combinatoire difficile à résoudre. Ce problème a été abordé avec le parti pris de se limiter à des modélisations simples, implémentables ex nihilo en moins de 100 d'heures d'ingénierie, en utilisant la technologie SAT. Quatre encodages CNF ont été essayés. Le plus efficace s'est avéré être celui qui produit les plus petites formules, bien que ce soit le seul qui ne permette pas à la propagation unitaire de restaurer la cohérence de domaine sur les contraintes de concordance de motifs.

Mots-clés

Concordance de motifs, encodage CNF, SAT, CSP

Abstract

Edge-matching puzzles — of which Eternity II (unsolved since 2007) is an iconic example — are the archetype of the notoriously difficult combinatorial problem. This problem was tackled with the aim of restricting the models to simple ones, implementable from scratch in less than 100 engineering hours, using SAT technology. Four CNF encodings were attempted. The most effective turned out to be the one that produced the smallest formulas, although it is the only one that does not allow unit propagation to restore domain consistency on the pattern-matching constraints.

Keywords

Edge Matching Problems, CNF encoding, SAT, CSP

1 Introduction

Le problème "Edge Matching Puzzle", popularisé par la commercialisation du jeu Eternity II en 2007, constitue un terrain idéal pour challenger les techniques les plus efficaces de résolution de problèmes combinatoires. Les instances basées sur des grilles de dimensions supérieures à 10×10 sont de véritables challenges pour les solveurs SAT les plus performants, sans compter que le puzzle original de 2007, de dimension 16 par 16, est encore irrésolu aujourd'hui.

Quatre encodages CNF ont été essayés. Ils se différencient par la modélisation des contraintes de concordance de motifs, la manière de les encoder en clauses proposition-

nelles, et les capacités déductives de la propagation unitaire appliquée aux formules produites. Il s'avère que le codage le plus performant est celui qui propage le moins, mais qui produit les plus petites formules, ce qui confirme qu'il est hasardeux de choisir à priori un encodage CNF de contraintes énumératives sur le seul critère de sa capacité à permettre à la propagation unitaire de restaurer la cohérence de domaine, que nous appellerons *DC-compliance* dans la suite de l'article. Une telle observation a déjà été faite, notamment dans [16] et [15], mais entre des représentations respectivement linéaires et logarithmiques des valeurs des variables, alors que les encodages comparés dans cet article sont tous basés sur des représentations linéaires.

Le puzzle Eternity II est constitué d'une grille carrée de 16×16 et 256 pièces carrées ayant des motifs sur chacun de leurs 4 côtés. Un motif spécifique, de couleur grise, indique que le côté concerné doit être situé en bordure de la grille. Quatre pièces de coin possèdent chacune deux côtés gris. Cinquante-six pièces de bords possèdent chacune un côté gris. Les 196 autres pièces, dites centrales, ne possèdent aucun côté gris. Le but du jeu est de choisir, pour chaque pièce, une position et une orientation telles que chaque bord gris se situe en bordure de la grille (on dira que les contraintes de bords sont respectées) et que chaque paire de pièces adjacentes aient les mêmes motifs sur leurs côtés contigus (on dira que les contraintes de concordance de motifs sont respectées). Deux millions de dollars étaient promis à la première personne trouvant une solution avant le 31 décembre 2008. Cette date limite a été repoussée au 31 décembre 2009, puis 2010, mais aucune solution n'a encore été exhibée au moment où je rédige cet article, en février 2025.

Le puzzle Eternity II est une instance d'un problème NP-complet [7], qui a inspiré plusieurs travaux dans le milieu de la recherche en informatique et au-delà, visant essentiellement à maximiser le nombre de concordances de motifs de l'instance originale et des variantes dont les plus connues, de dimensions 10×10 , 12×12 , 14×14 , 16×16 ont été publiées à l'occasion du "Meta 2010 Eternity II contest" [9]. Ces instances sont challengées comme des problèmes d'optimisation, notamment dans [13, 14, 12].

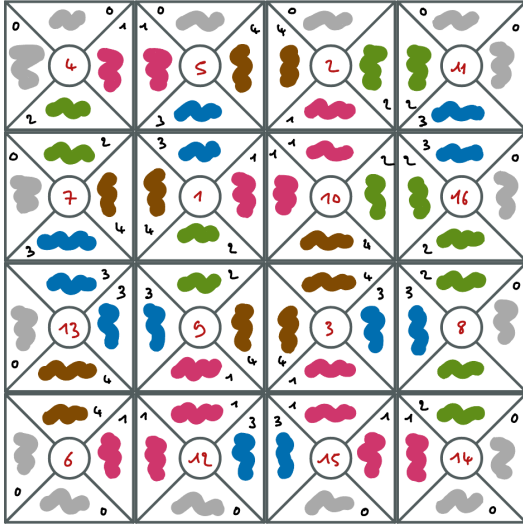


FIGURE 1 – Une instance résolue de Edge-Matching problem.

2 Résolution avec un solveur SAT

Les quatre encodages CNF ayant été essayés pour la résolution du problème à l'aide d'un solveur SAT reposent sur quatre modélisations de plus haut niveau sous forme de réseaux de contraintes à domaines finis. Dans cette section, ces quatre modèles sont d'abord présentés, puis leur traduction en clauses propositionnelles sont ensuite explicitées. Enfin, les capacités de déduction de la propagation unitaire appliquée à ces encodages sont comparées.

On notera n le nombre de lignes et de colonnes de la grille, dont chaque case sera identifiée par ses coordonnées (i, j) avec $i, j \in \{0, \dots, n-1\}$. On notera m le nombre de motifs possibles pour les côtés des pièces, sans comptabiliser le motif de bordure de grille. Les motifs seront identifiés par les entiers 0 à m , 0 désignant le motif de bordure. Les pièces sont numérotées de 0 à $n^2 - 1$.

2.1 Modélisation à l'aide de contraintes

Les noms des quatre modèles présentés indiquent la nature des contraintes de concordance qui imposent que les motifs des côtés en contact de deux pièces situées dans des cases contiguës soient identiques. FULL exprime que chaque situation d'adjacence est représentée par une seule contrainte énumérative, HALF exprime que chaque contrainte de concordance est décomposée en deux demi-contraintes connectées par des variables représentant le motif des deux bords en contact. Le chiffre qui suit indique l'arité des contraintes énumératives.

2.1.1 Modèle FULL-4

À chaque case de la grille de coordonnées (i, j) sont associées deux variables :

- Une variable $P_{i,j}$ de domaine $\{0, \dots, n^2 - 1\}$, dont la valeur indique la pièce présente dans la case.
- Une variable $O_{i,j}$ de domaine $\{0, \dots, 3\}$, dont la

valeur indique l'orientation de la pièce présente dans la case.

Les contraintes sont les suivantes :

1. Chaque pièce occupe exactement une case : $\text{alldiff}(P_{i,j})$.
2. À chaque paire $(i, j), (k, l)$ de cases contiguës, verticalement ou horizontalement, on associe une contrainte de concordance reliant les variables $P_{i,j}, O_{i,j}, P_{k,l}$ et $O_{k,l}$. Cette contrainte est définie en extension. Elle exprime les combinaisons (pièce, orientation) qui assignent le même motif aux côtés en contact.

2.1.2 Modèle FULL-2

Ce modèle est utilisé dans [1]. On reprend le principe de FULL-4, mais en fusionnant les variables indiquant la pièce située dans chaque case et son orientation, de manière à rendre binaires les contraintes de concordance. À chaque case (i, j) est donc associée une seule variable $Q_{i,j}$ de domaine $0 \dots 4n^2 - 1$ dont chaque valeur représente une pièce et une des orientations possibles de cette pièce.

Les contraintes sont les suivantes :

1. Chaque pièce occupe exactement une case.
2. À chaque paire $(i, j), (k, l)$ de cases contiguës, verticalement ou horizontalement, on associe une contrainte de concordance reliant les variables $Q_{i,j}$ et $Q_{k,l}$. Cette contrainte est définie en extension. Elle exprime les combinaisons (pièce, orientation) qui assignent le même motif aux côtés en contact.

2.1.3 Modèle HALF-3

Pour chaque case (i, j) , on reprend les variables $P_{i,j}, O_{i,j}$ et la contrainte $\text{alldiff}(P_{i,j})$ du modèle FULL-4 et on introduit des variables supplémentaires à domaines $0 \dots m$ représentées par les symboles $U_{i,j}$ (pour up), $R_{i,j}$ (pour right), $D_{i,j}$ (pour down) et $L_{i,j}$ (pour left). Les symboles associés à deux bords en contact représentent une même variable. Par exemple, $R_{0,0}$ et $L_{0,1}$ représentent une même variable.

Pour chaque case (i, j) , 4 contraintes d'arité 3 définissent le motif de chaque côté :

1. Une contrainte reliant $P_{i,j}, O_{i,j}$ et $U_{i,j}$.
2. Une contrainte reliant $P_{i,j}, O_{i,j}$ et $R_{i,j}$.
3. Une contrainte reliant $P_{i,j}, O_{i,j}$ et $D_{i,j}$.
4. Une contrainte reliant $P_{i,j}, O_{i,j}$ et $L_{i,j}$.

Ces contraintes sont définies en extension.

Des contraintes supplémentaires restreignent les valeurs et orientations des pièces situées dans les cases de bords et de coins, ainsi que les valeurs des pièces situées dans les cases centrales.

Ce modèle est très proche de celui utilisé dans [6], dans lequel des variables distinctes sont utilisées pour représenter les motifs des 4 bords de chaque pièce, avec ajout de contraintes d'égalités entre bords contigus.

2.1.4 Modèle HALF-2

On reprend le principe HALF-3, mais en fusionnant les variables représentant la pièce située dans une case et son orientation selon le principe du modèle FULL-2. La figure 2 donne une vue synthétique des contraintes représentant les règles de concordance de motifs pour chacun des 4 modèles.

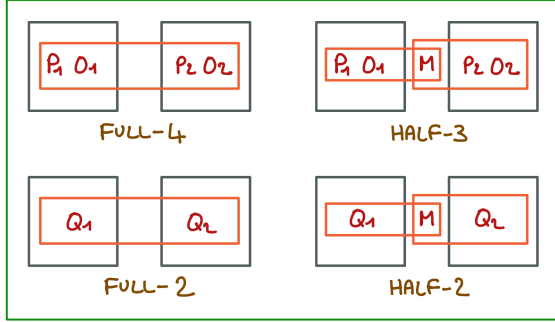


FIGURE 2 – Vue synthétique des contraintes exprimant les règles de concordance des motifs entre pièces adjacentes dans les 4 modèles proposés.

2.2 Puissance déductive de la propagation

2.2.1 Propagation de FULL-4 versus HALF-3

Avant de passer aux encodages CNF ayant été utilisés pour ces quatre modèles, il est intéressant de comparer les capacités de déduction de la restauration de la cohérence de domaines sur les contraintes utilisées, aussi appelée propagation.

Le premier point important est que la décomposition de chaque contrainte d'adjacence en deux demi-contraintes, avec introduction de variables auxiliaires, n'affecte pas les déductions réalisables par propagation. En effet, les mêmes informations peuvent être déduites par propagation des contraintes de concordance de FULL-4 et HALF-3.

Rappelons que chaque contrainte de concordance de motifs entre cases contiguës de FULL-4 est décomposée dans HALF-3 en deux contraintes reliées par une variable qui représente le motif commun aux deux côtés en contact des pièces concernées.

Soient deux cases contiguës C_1 et C_2 . Soient q la contrainte de concordance sur P_1, O_1, P_2, O_2 associée à la frontière commune entre ces deux cases. Soient q_1 sur P_1, O_1, M et q_2 sur P_2, O_2, M , les contraintes représentant la décomposition de q .

Soit une valeur $P_1 = p$ n'ayant pas de support pour q . Soit d l'ensemble des motifs appartenant à des supports de $P_1 = p$ pour q_1 . Le fait que $P_1 = p$ n'ait pas de support implique qu'aucune des assignations autorisées de P_2 et O_2 ne comporte un motif de d sur le côté de C_2 en contact avec C_1 . En conséquence, la propagation de q_2 retire ces motifs du domaine de M , ce qui permet à la propagation de q_1 de retirer p du domaine de P_1 . Symétriquement, toute valeur $P_2 = v$ n'ayant pas de support pour q est filtrée par la propagation de q_1 et q_2 .

Un raisonnement similaire, appliqué à une valeur $O_1 = r$ (respectivement $O_2 = r$) n'ayant pas de support pour q , montre que les propagations de q_1 et q_2 retirent r du domaine de O_1 (respectivement O_2).

2.2.2 Impact de la fusion des variables dans FULL-2 et HALF-2

Les deux variables de HALF-3 représentant respectivement la pièce située dans une case et son orientation sont fusionnées en une seule dans HALF-2, dans le but de représenter les règles de concordance par des contraintes binaires. Cette fusion permet de réaliser toutes les propagations supportées par HALF-3, *et plus encore*, car certaines déductions pouvant être inférées par restauration de la cohérence de domaine sur les demi contraintes de concordance binaires ne sont *pas* représentables par des sous-domaines des variables utilisées par les demi contraintes de concordance d'arité 3 du modèle HALF-3.

Plus précisément, certaines assignations partielles des variables fusionnées de HALF-2 permettent de représenter des situations qui ne sont pas représentables par des assignations partielles de variables de HALF-3. Par exemple, pour un puzzle de 4×4 , l'assignation $Q_{2,2} \in \{19, 33\}$ de HALF-2, qui exprime que la case (2, 2) peut être occupée uniquement par la pièce 3 en orientation 1 ou par la pièce 1 en orientation 2, ne peut être représentée par aucune assignation partielle de HALF-3. L'assignation partielle de HALF-3 la plus restrictive couvrant ces deux possibilités est $P_{2,2} \in \{1, 3\}, O_{2,2} \in \{1, 2\}$, qui donne moins d'information, puisqu'elle couvre quatre configurations (pièce 1 en orientations 1 et 2 et pièce 3 en orientations 1 et 2) au lieu de deux.

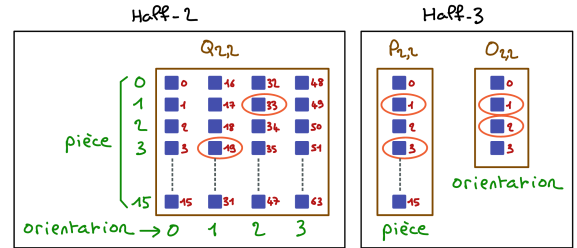


FIGURE 3 – Une assignation partielle de la variable de fusion $Q_{2,2}$ du modèle HALF-2 et l'assignation partielle la plus restrictive des variables correspondantes du modèle HALF-3 qui couvre les mêmes configurations (pièce, orientation).

Le même raisonnement s'applique à la fusion des variables représentant les pièces et orientations permettant de passer du modèle FULL-4 au modèle FULL-2.

En résumé, le modèle HALF-2 permet de réaliser plus de propagations que le modèle HALF-3 parce qu'il exploite des représentations plus expressives des combinaisons (pièce, orientation). Pour la même raison, le modèle FULL-2 permet de réaliser plus de propagations que le modèle FULL-4. Les modèles HALF-3 et FULL-4 permettent de réaliser les mêmes propagations sur les variables repré-

sentant les pièces et leurs orientation, mais HALF-3 permet de réaliser en plus des propagations prenant en compte les variables de motifs, qui n'existent pas dans FULL-4. La même remarque s'applique à HALF-2 et FULL-2. Par ordre décroissant de puissance de propagation, on a donc HALF-2, puis HALF-3 et FULL-2, et enfin FULL-4. HALF-3 et FULL-2 sont difficiles à comparer car HALF-3 tire parti des variables de motifs et peut permettre à un solveur de faire des hypothèses sur ces variables, alors que FULL-2 n'a pas cet avantage, mais permet des représentations plus fines des combinaisons (pièce, orientation).

2.3 Encodage des contraintes en CNF

2.3.1 Contraintes d'unicité

Chaque variable initiale V de domaine $a \dots b$ est traduite par n variables propositionnelles qui seront notées $V = a, \dots, V = b$. Un ensemble de clauses exprime l'unicité de la valeur de V en utilisant l'encodage proposé dans [11], de taille linéaire, avec un nombre linéaire de variables auxiliaires. Une telle contrainte peut être encodée sans variables auxiliaires avec un nombre quadratique de clauses, mais des essais réalisés avec les instances de problèmes utilisés pour comparer les différents encodages ont montré que cette option est moins efficace.

La contrainte alldiff des modèles FULL-4 et HALF-3 est spécifiquement une contrainte de permutation. Elle est décomposée en deux contraintes d'unicité : une qui exprime que chaque case contient exactement une pièce, et une qui exprime que chaque pièce occupe exactement une case. La première contrainte est déjà nécessaire pour encoder l'unicité de la valeur assignée à chaque variable. La deuxième est encodée de la même manière, avec l'approche présentée dans [11]. Une telle décomposition ne permet pas à la propagation unitaire du solveur SAT de restaurer la cohérence de domaine du alldiff, mais aucun encodage ne le permet avec un nombre polynomial de clauses [4].

Dans les modèles FULL-2 et HALF-2, la nécessité que chaque pièce $p \in 0 \dots n^2 - 1$ occupe exactement une case est encodée par une contrainte d'unicité entre les $4n^2$ variables propositionnelles $Q_{i,j} = rn^2 + p$ pour $i, j \in \{0, \dots, n-1\}$ et $r \in \{0, \dots, 3\}$, qui exprime qu'une case et une seule contient la pièce p dans une et une seule orientation r .

2.3.2 Contraintes énumératives

Les contraintes de concordance de motifs entre pièces adjacentes se présentent sous forme énumérative dans les quatre modèles proposés, mais des approches différentes ont été adoptées pour les encoder en CNF.

Les contraintes d'adjacence du modèle FULL-4 ont été encodées avec la méthode introduite dans [2], qui est une généralisation du « support encoding » introduit dans [8] dans le cadre restreint des contraintes binaires. Cette approche encode en CNF chaque propagation possible de la contrainte source en introduisant une variable auxiliaire pour chaque support. Le nombre de clauses produit est en $\Theta(km)$, avec k l'arité de la contrainte source et m son nombre de solutions.

Les contraintes d'adjacence binaires des modèles FULL-2

et HALF-2 ont été encodées avec la technique « support encoding » introduite dans [8], qui permet à la propagation unitaire de restaurer la cohérence de domaine de la contrainte source sans ajout de variables auxiliaires, avec un nombre de clauses proportionnel au nombre de solutions de la contrainte source.

Dans le cas du modèle HALF-3, dans lequel les règles d'adjacence sont décomposées en deux contraintes ternaires partageant une même variable auxiliaire, le choix d'un encodage plus concis a été fait : pour chaque case (i, j) , chaque pièce p et chaque orientation r , et chaque variable C associée à un côté de la case, la clause

$$\neg(P_{i,j} = p) \vee \neg(O_{i,j} = r) \vee (C = m)$$

est produite, où m est le motif placé sur le côté désigné par C lorsque la pièce p a l'orientation r .

En complément, des clauses sont ajoutées pour encoder les propriétés suivantes (qui sont prises en compte implicitement dans les autres modèles) :

1. Une case de coin ne peut contenir qu'une pièce de coin, dans une seule orientation possible.
2. Une case de bord ne peut contenir qu'une pièce de bord, avec une seule orientation possible.
3. Une case de centre ne peut contenir qu'une pièce de centre.

Cet encodage ne permet pas à la propagation unitaire de restaurer complètement la cohérence de domaine des contraintes de concordance de motifs. Il ne permet même pas à la propagation unitaire de détecter certaines incohérences. Par exemple, supposons que le domaine de la variable $P_{1,1}$ contient uniquement des pièces dont les côtés ne sont ni rouge, ni vert, que le domaine de la variable $P_{1,2}$ contient uniquement des motifs dont les côtés sont rouges ou verts, et que le domaine de la variable $R_{1,1}$, aussi nommée $L_{1,2}$, qui représente le motif de l'arête commune des cases $(1, 1)$ et $(1, 2)$, contienne toutes ses valeurs initiales. Une restauration de la cohérence de domaines des contraintes énumératives sous-jacentes aurait pour effet de propager $R_{1,1} \neq \text{rouge}$ et $R_{1,1} \neq \text{vert}$, puis de retirer toutes les valeurs du domaine de $P_{1,2}$. Mais les clauses de l'encodage HALF-3 ne permettent pas ces propagations.

2.4 Comparaison synthétique des quatre encodages CNF

Voici un bref résumé des points faibles et des points forts des 4 encodages CNF.

Les encodages FULL-2 et HALF-2 sont ceux qui permettent à la propagation unitaire de réaliser le plus de déductions, d'une part grâce à une représentation des combinaisons (pièces, orientations) qui maximise la quantité d'information représentable par les assignations partielles, et d'autre part en réalisant toutes les inférences relevant de la restauration de la cohérence de domaine. Le modèle HALF-2 a l'avantage de produire des formules de plus petite taille. Le modèle FULL-4 utilise une représentation moins expressive que FULL-2 et HALF-2. Son encodage CNF permet à

la propagation unitaire de restaurer complètement la cohérence de domaine des contraintes de concordance de motifs, mais au prix de l'ajout de variables auxiliaires qui ne sont pas nécessaires dans les représentations CNF de FULL-2 et HALF-2. De surcroît, c'est celui des quatre encodages implémentés qui produit les plus grosses formules. Il ne présente à priori aucun avantage par rapport à FULL-2-CNF et HALF-2-CNF.

La représentation adoptée pour encoder HALF-3 en CNF est celle qui propage le moins. C'est la conséquence du choix qui a été fait de privilégier la compacité des formules produites au détriment des déductions réalisables par propagation unitaire. "Sur le papier", le seul intérêt de l'encodage CNF choisi pour le modèle HALF-3 est donc de produire de plus petites formules que tous les autres encodages, mais sans toutefois changer l'ordre de grandeur de leurs tailles. Selon des critères basés sur les capacités de propagation, on pourrait considérer un tel choix comme une erreur de conception. Et pourtant, les résultats présentés dans la section suivante montrent que c'est l'encodage le plus efficace !

3 Résultats

Les différentes approches de résolution ont été appliquées à des lots de 100 instances de puzzle de tailles 5×5 à 10×10 générées aléatoirement, avec une limite de temps de 1000 secondes pour chaque instance, sur un ordinateur MacBook M1 Pro. Un vérificateur a été utilisé pour valider toutes les solutions trouvées.

La production d'un puzzle aléatoire de $n \times n$ avec m motifs différents se fait en trois étapes :

1. Production d'une solution d'un puzzle torique (donc sans bordure, le bord bas d'une pièce située sur la dernière ligne est en contact avec le bord haut de la pièce située sur la première ligne dans la même colonne, le bord droit d'une pièce située sur la dernière colonne est en contact avec le bord gauche de la pièce de la même ligne située sur la première colonne) avec équilibrage des nombres d'occurrences de motifs identiques, au sens où chaque motif apparaît $2n^2/m$ fois ou $1 + 2n^2/m$ fois sur les $2n^2$ frontières entre pièces contiguës.
2. Assignment du motif 0 à chacun des $4n$ bords de la grille.
3. Mélange des pièces ainsi obtenues.

Les résultats obtenus avec les quatre encodages CNF et la version 4.0.1 du solveur kissat [5] (en utilisant les paramètres par défaut) sont présentés dans les tableaux 4, 5, 6, 7 du plus performant au moins performant. Les temps moyens de résolution ont été calculés uniquement sur les instances résolues.

Pour les quatre encodages, la dispersion des temps de résolution entre puzzles de même taille est très élevée. Par exemple, dans le lot de puzzles de 8×8 , avec l'encodage le plus performant, on constate un facteur 100 entre les temps de résolution minimaux et maximaux. Les 6 puzzles de 9×9

résolus le plus rapidement (moins de 63 secondes) sont traités plus rapidement que les 6 puzzles de 8×8 dont la résolution prend le plus de temps (plus de 63 secondes).

La mauvaise performance du FULL-4 montre que la DC-compliance ne garantit pas l'efficacité d'un encodage CNF de contraintes énumératives, en particulier lorsqu'il introduit des variables impliquées.

L'efficacité relative de HALF-3 par rapport à HALF-2 est surprenante, car le second permet toutes les propagations possibles de chaque contrainte de concordance de motifs, alors que le premier ne permet même pas à la propagation unitaire de détecter les incohérences sur ces contraintes, qui sont par ailleurs décomposées de la même manière dans les modèles CSP sous-jacents. En outre, les formules produites par HALF-2 ne sont que deux fois plus grosses que celles produites par HALF-3.

Les performances relativement proches de FULL-2 et HALF-2 n'appellent pas de commentaire.

3.1 Conclusion et perspectives

La résolution du problème de recherche de type « Edge matching puzzle », connu sous le nom commercial « Eternity II », a été abordée avec quatre encodages CNF et un solveur SAT « Up-to-date ».

Les modèles utilisés sont directement inspirés de la spécification native du problème, reposant sur des contraintes de concordance de motifs sur les paires de cases contiguës. Bien que la littérature propose des modèles plus sophistiqués, impliquant des contraintes sur un nombre plus élevé de cases [10, 3], elles n'ont pas été exploitées ici. En effet, l'objectif de ce travail n'était pas de relever le défi particulier posé par ce problème, mais plutôt de le traiter comme un exemple type de problème combinatoire difficile se présentant sous la forme d'un réseau relativement homogène de contraintes de même nature et de même taille.

Le codage CNF qui s'est avéré le plus efficace est celui qui produit les plus petites formules, mais au prix d'une réduction drastique des capacités de déduction de la résolution unitaire, puisque les propagations des contraintes de concordance de motifs ne peuvent se faire que lorsque la pièce située dans une case et son orientation sont complètement déterminées.

Ce constat nous rappelle que la résolution des problèmes NP-complets relève encore aujourd'hui d'une démarche essentiellement expérimentale, et que nous sommes loin de comprendre parfaitement les critères d'efficacité des solveurs CDCL. Le cas du codage HALF-3 mérite une étude plus approfondie. Pourquoi est-il plus efficace qu'un codage produisant des formules de taille seulement deux fois plus grande tout en permettant plus de propagations ? Est-ce que les propagations supplémentaires sont inutiles pour la recherche d'une solution ? (Le seraient-elles pour une preuve d'incohérence ?) Ou bien les clauses apprises par le solveur permettent-elles de réaliser certaines propagations qui ne sont pas initialement implémentées dans l'encodage ?

Il semble donc utile d'étudier plus en détail le comportement des solveurs CDCL sur des formules CNF résultant

Quel codage CNF pour les contraintes de concordance de motifs d'Eternity II ?

d'un encodage de contraintes, et en particulier l'hypothèse que ces solveurs puissent compenser des déficits de propagation en produisant des clauses permettant de réaliser des propagations qui ne sont pas initialement supportées.

taille	# variables	# clauses	poids	% résolu	temps moyen
5 × 5	2.6k	15k	0.2 Mo	100	0.03
6 × 6	5.2k	31k	0.5 Mo	100	0.18
7 × 7	9.1k	57k	1 Mo	100	1.39
8 × 8	15k	97k	1.7 Mo	100	20.2
9 × 9	24k	154k	2.7 Mo	49	380
10 × 10	35k	233k	4.5 Mo	0	—

FIGURE 4 – Résultats de l’encodage **HALF-3**

taille	# variables	# clauses	poids	% résolu	temps moyen
5 × 5	7.5k	23k	0.3 Mo	100	0.00
6 × 6	15k	48k	0.8 Mo	100	0.09
7 × 7	29k	90k	1.8 Mo	100	1.04
8 × 8	49k	155k	3.8 Mo	100	27.8
9 × 9	79k	250k	7.6 Mo	30	425
10 × 10	120k	383k	14 Mo	0	—

FIGURE 5 – Résultats de l’encodage **FULL-2**

taille	# variables	# clauses	poids	% résolu	temps moyen
5 × 5	8k	28k	0.4 Mo	100	0.00
6 × 6	17k	58k	0.9 Mo	100	0.03
7 × 7	30k	107k	1.7 Mo	100	0,54
8 × 8	51k	180k	3.1 Mo	100	28.6
9 × 9	82k	286k	5 Mo	25	530
10 × 10	124k	433k	7.8 Mo	0	—

FIGURE 6 – Résultats de l’encodage **HALF-2**

taille	# variables	# clauses	poids	% résolu	temps moyen
5 × 5	21k	64k	0.7 Mo	100	0.34
6 × 6	80k	230k	3.5 Mo	100	2.68
7 × 7	246k	736k	12 Mo	100	47.4
8 × 8	672k	2010k	33 Mo	37	497
9 × 9	1470k	4420k	75 Mo	0	—

FIGURE 7 – Résultats de l’encodage **FULL-4**

Références

- [1] Carlos Ansótegui, Ramón Béjar, César Fernández, and Carles Mateu. How hard is a commercial puzzle : the eternity ii challenge. In *Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*, volume 184, 01 2008.
- [2] Fahiem Bacchus. Gac via unit propagation. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages 133–147, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- [3] Thierry Benoist. Fast global filtering for eternity ii. *Constraint Programming Letters*, 3 :36–49, 01 2008.
- [4] Christian Bessiere, George Katsirelos, Nina Nardyska, and Toby Walsh. Circuit complexity and decompositions of global constraints. *IJCAI International Joint Conference on Artificial Intelligence*, 05 2009.
- [5] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froyls, and Florian Pollitt. CaDiCaL, Gimsat, IsaSAT and Kissat entering the SAT Competition 2024. In Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, volume B-2024-1 of *Department of Computer Science Report Series B*, pages 8–10. University of Helsinki, 2024.
- [6] Joffrey Cuvillier and Rémi Szymkowiak. Résolution du jeu eternity 2 avec les technologies sat, 2010.
- [7] Erik Demaine and Martin Demaine. Jigsaw puzzles, edge matching, and polyomino packing : Connections and complexity. *Graphs and Combinatorics*, 23 :195–208, 06 2007.
- [8] Ian Gent. Arc consistency in sat. In *Proceedings of the 15th European Conference on Artificial Intelligence, ECAI'2002*, pages 121–125, 01 2002.
- [9] International conference on metaheuristics and nature inspired computing (meta), 2010.
- [10] Fabio Salassa, Wim Vancroonenburg, Tony Wauters, Federico Della Croce, and Greet Vanden Berghe. Milp and max-clique based heuristics for the eternity ii puzzle, 09 2017.
- [11] Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, pages 827–831, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- [12] Wim Vancroonenburg, Tony Wauters, and Greet Vanden Berghe. A two phase hyper-heuristic approach for solving the eternity ii puzzle. In *Conference : 3rd International Conference on Metaheuristics and Nature Inspired Computing (META 2010)*, 10 2010.
- [13] Wei-Sin Wang and Tsung-Che Chiang. Solving eternity-ii puzzles with a tabu search algorithm. In *Proceedings of International Conference on Metaheuristics and Nature Inspired Computing, 2010*, 10 2010.
- [14] Tony Wauters, Wim Vancroonenburg, and Greet Vanden Berghe. A guide-and-observe hyper-heuristic approach to the eternity ii puzzle. *Journal of Mathematical Modelling and Algorithms*, 11 :217–233, 09 2012.
- [15] Neng-Fa Zhou. Yet another comparison of sat encodings for the at-most-k constraint, 2020.
- [16] Neng-Fa Zhou and Håkan Kjellerstrand. Optimizing sat encodings for arithmetic constraints. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming*, pages 671–686, Cham, 2017. Springer International Publishing.

Session 2, lundi 30 juin, 15h00-16h00 : Applications

Sélection quasi-optimale de routes pour la surveillance de réseaux à l'aide de la génération de colonnes

Alice Burlats, Cristel Pelsser, Pierre Schaus

UCLouvain, ICTeam

2025

Résumé

Un système de gestion de réseau robuste doit permettre de détecter et de diagnostiquer rapidement les pannes. Avec la tomographie booléenne, la surveillance d'un ensemble suffisant de routes permet de rendre toute panne détectable et identifiable de manière unique (1-identifiable). Étant donné que ces sondes génèrent du trafic dans le réseau, il est crucial de sonder le plus petit ensemble de routes possible tout en garantissant une supervision efficace. Pour ce problème nous expérimentons une approche basée sur la génération de colonnes pour résoudre les problèmes de mémoire rencontrés par un modèle de programmation linéaire en nombres entiers tout en retournant de meilleures solutions qu'un algorithme glouton.

Mots-clés

Tomographie booléenne, supervision de réseaux, programmation linéaire en nombres entiers, génération de colonnes

1 Introduction

Les pannes dans les réseaux informatiques sont fréquentes. Elles surviennent suite à des erreurs de configurations, des câbles endommagés, des phénomènes naturels ou simplement l'usure. La tomographie booléenne consiste à déterminer l'état de chaque nœud à l'aide de mesures et de règles d'inférence. Lorsqu'un nœud tombe en panne, toutes les routes le traversant tombent également en panne. Une sonde (un paquet ICMP) peut donc être envoyée le long d'une route pour vérifier si elle atteint sa destination, et donc s'assurer que la route, et tous les nœuds qu'elle traverse, est fonctionnelle. En sondant régulièrement un ensemble suffisant de routes de mesure, où chaque nœud est traversé par au moins une route, toute panne dans le réseau peut être détectée.

Au-delà de la détection des défaillances, il est possible de les localiser. L'ensemble des routes de mesures qui traversent un nœud constitue son *symptôme*, ce qui signifie que sa défaillance entraînera la coupure de toutes ces routes. La défaillance d'un seul nœud peut être localisée si chaque nœud présente un symptôme unique. Cette condition est connue sous le nom de 1-identifiabilité.

La Figure 1 présente une petite topologie de réseau. Les styles de lignes et les couleurs représentent les différentes routes de mesure. Dans (1), chaque nœud est traversé par

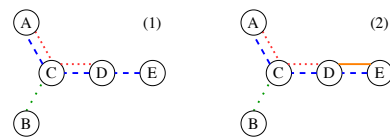


FIGURE 1 – Illustration de la 1-identifiabilité. Avec les routes mises en évidence : (1) tous les nœuds sont couverts mais pas 1-identifiables; (2) tous les nœuds sont 1-identifiables.

au moins une route de mesure. Ainsi, lorsque l'un d'entre eux rencontre une défaillance, il est possible de la détecter. Mais A et D ont la même signature. Si l'un ou l'autre tombe en panne, dans les deux cas la route liant A à E et A à D ne fonctionne plus. La panne sera détectée mais il sera impossible de déterminer sans ambiguïté sur quel nœud elle se situe. Le réseau est couvert mais les défaillances ne sont pas 1-identifiables. Dans (2), une route liant E à D a été ajoutée. Avec cette route supplémentaire, le réseau est 1-identifiable car chaque nœud est traversé par un ensemble unique de routes de mesure.

Être réactif aux défaillances implique d'envoyer des sondes très fréquemment, ce qui peut perturber le réseau. Ainsi, pour limiter ce problème, il est nécessaire de minimiser le nombre de routes de mesures qui doivent être sondées.

Dans notre étude, nous considérons que nous n'avons pas le contrôle sur la définition des routes dans le réseau : le protocole de routage offre une route entre chaque paire de nœuds. Nous piochons un sous-ensemble de routes de mesure parmi ces routes pour mesurer le réseau.

Nos contributions peuvent être résumées comme suit :

- Nous formulons le problème de 1-identifiabilité par ensembles (**1IE**) (Section 3);
- Nous présentons un algorithme glouton et un modèle de programmation linéaire en nombres entiers (PLNE) pour résoudre ce problème (Sections 3.1 et 3.2);
- Nous proposons une approche basée sur la génération de colonnes (GC) pour résoudre un modèle utilisant moins de variables (Section 3.3);
- Nous montrons que la génération de colonnes est capable de trouver de meilleures solutions que le glouton tout en utilisant moins de 25% des variables du modèle PLNE complet. (Section 4).

2 Travaux antérieurs

Plusieurs approches pour la sélection de routes de mesure dans un réseau ont été explorées. Stanic et al. [11] présentent un algorithme glouton et un modèle PLNE. L'algorithme glouton commence par sélectionner toutes les routes possibles, puis les supprime une par une jusqu'à ce que la suppression d'une route supplémentaire entraîne une rupture de la couverture ou une violation de la propriété de 1-identifiabilité. Cette méthode est assez inefficace en termes de complexité temporelle car le nombre de routes nécessaires à la supervision est faible, comme montré dans la section 4. La conception de leur modèle PLNE est similaire à celle du nôtre. Pepe et al. [8] conçoivent également un algorithme glouton pour garantir la 1-identifiabilité des liens dans le réseau. Leur algorithme produit une solution avec autant de routes que d'éléments à identifier, ce qui est sans doute sous-optimal.

Plusieurs études [1, 2, 7, 9] traitent d'un problème connexe qui implique la sélection de moniteurs à partir des nœuds du réseau. Dans cette approche, tout chemin reliant deux moniteurs sélectionnés est automatiquement considéré comme un chemin de surveillance, ce qui élimine le besoin de choisir explicitement les routes. Cette approche conduit à la surveillance d'un plus grand nombre de routes.

3 Définition et approches

Le problème de 1-identifiabilité par ensemble (IIE) : Soit un univers U et une collection d'ensembles $S = \{S_1, \dots, S_{|S|}\}$, où $\forall S_i \in S, S_i \subseteq U$. Nous notons $S^i \subseteq S$ la collection contenant tous les ensembles contenant l'élément $i, \forall i \in U$. Le but de ce problème est de trouver le sous-ensemble $S^* \subseteq S$ de cardinalité minimale tel que :

- $\forall i \in U, S^i \cap S^* \neq \emptyset$ (couverture);
- $\forall i \in U, S^i \cap S^*$ est unique (1-identifiabilité).

Pour chaque nœud $i \in U$, nous appelons l'intersection $S^i \cap S^*$ son *symptôme*. Le but du IIE est de sélectionner la plus petite collection d'ensembles tel que chaque élément d'un univers dispose d'un symptôme unique.

Nous pouvons modéliser notre problème de sélection de routes comme un IIE : L'univers U représente l'ensemble de nœuds de la topologie, il contient un élément pour chaque nœud présent dans le réseau. Chaque route dans le réseau est représentée comme un ensemble non-ordonné contenant chacun des nœuds traversés par la route. Ainsi, la collection S représente l'ensemble des routes définies par le protocole de routage.

3.1 Algorithme glouton

Nous proposons l'algorithme 1, inspiré de l'algorithme MNMP [7], mais adapté pour travailler sur la sélection d'un ensemble de routes de mesure, plutôt qu'un ensemble de nœuds moniteurs.

Chaque ensemble S_i de la collection des ensembles candidats S est représenté par deux bitsets. Le premier, $\Psi(S_i)$, représente l'ensemble des éléments couverts par S_i . Il s'agit

Algorithm 1 Algorithme glouton

```

1:  $S^* \leftarrow \{\}$  ▷ Ensemble de routes de mesures
2:  $I \leftarrow U$  ▷ Nœuds non couverts
3:  $I' \leftarrow \{(i, j) \in U^2 \mid i < j\}$  ▷ Paires non distinguées
4: while  $I' \neq \emptyset$  do
5:    $S_{\text{choisi}} \leftarrow \arg \max_{S_i \in S \setminus S^*} |I' \cap \Omega(S_i)|$ 
6:    $S^* \leftarrow S^* \cup \{S_{\text{choisi}}\}$ 
7:    $I' \leftarrow I' \setminus \Omega(S_{\text{choisi}})$ 
8:    $I \leftarrow I \setminus \Psi(S_{\text{choisi}})$ 
9: end while
10: if  $I \neq \emptyset$  then ▷ Au plus un nœud non couvert
11:    $S_{\text{choisi}} \leftarrow \arg \max_{S_i \in S \setminus S^*} |I \cap \Psi(S_i)|$ 
12:    $S^* \leftarrow S^* \cup \{S_{\text{choisi}}\}$ 
13: end if
14: return  $S^*$ 

```

d'un bitset de longueur $|U|$, où le bit i' est vrai ssi l'ensemble S_i contient i' . Pour chaque paire d'éléments i', j' , si l'ensemble S_i contient soit i' soit j' mais pas les deux, alors choisir cet ensemble assure que i' et j' auront un symptôme différent. On dit alors que S_i *distingue* i' et j' . Pour chaque $S_i \in S$, $\Omega(S_i)$ représente l'ensemble des paires d'éléments distinguées par S_i . C'est un bitset de longueur $\frac{|U| \cdot (|U|-1)}{2}$, contenant un bit pour chaque paire d'éléments qu'il est possible de constituer dans U , où le bit lié à la paire (i', j') est vrai ssi l'ensemble S_i permet de les distinguer.

La boucle *while* des lignes 4-9 est exécutée jusqu'à ce que chaque paire d'éléments puisse être distinguée. À chaque itération, l'ensemble permettant de distinguer le plus de paires non distinguées est ajouté. En cas d'égalité, nous sélectionnons au hasard un ensemble parmi les meilleurs. À la fin de cette boucle, il est possible qu'un élément unique ne soit pas couvert. En effet s'il est le seul à avoir un symptôme unique, alors il est distingué de tous les autres nœuds. Dans ce cas, un ensemble couvrant l'élément non couvert est arbitrairement ajouté.

3.2 Formulation PLNE

Nous utilisons un ensemble de variables binaires x . Nous avons une variable x_s par ensemble candidat $S_s \in S$, qui vaut 1 ssi l'ensemble S_s est sélectionné. Pour chaque élément $i \in U$, la collection des ensembles candidats qui couvrent i est représenté par un vecteur binaire $C_i = \{C_{i1}, \dots, C_{i|S|}\}$ de taille $|S|$ où C_{is} vaut 1 ssi l'ensemble S_s contient l'élément i . Pour chaque paire d'éléments $i < j \in U^2$, il existe également un vecteur binaire $D_{(ij)} = \{D_{(ij)1}, \dots, D_{(ij)|S|}\}$ de taille $|S|$ représentant la collection des ensembles candidats qui contiennent soit i , soit j , mais pas les deux. Il contient la collection d'ensembles qui distinguent i et j ; si au moins l'un d'entre eux est sélectionné, les symptômes des deux éléments sont différents.

Le modèle PLNE est composé de plusieurs contraintes. L'équation (1) exprime la fonction objectif, c'est-à-dire le nombre d'ensembles sélectionnés. L'équation (2) garantit que chaque élément est contenu dans au moins un des ensembles sélectionnés, i.e., que chaque élément est couvert. L'équation (3) est utilisée pour la 1-identifiabilité : elle im-

pose que, pour chaque paire d'éléments, il existe au moins un ensemble qui contient l'un d'entre eux et pas l'autre, c'est-à-dire qu'ils aient des symptômes différents.

$$\text{minimiser } \sum_{S_s \in S} x_s \quad (1)$$

tel que :

$$\sum_{S_s \in S} C_{is} x_s \geq 1 \quad \forall i \in U \quad (2)$$

$$\sum_{S_s \in S} D_{(ij)s} x_s \geq 1 \quad \forall (i, j) \in U^2, i < j \quad (3)$$

$$x_s \in \{0, 1\} \quad \forall S_s \in S \quad (4)$$

3.3 Génération de colonnes

Pour résoudre les problèmes de mémoire rencontrés avec le modèle PLNE (voir section 4) sur les grandes instances, nous explorons la GC pour n'utiliser qu'un petit sous-ensemble des variables du modèle original et limiter ainsi la consommation de mémoire.

Problème maître restreint : La GC commence par la résolution d'une relaxation linéaire réduite de notre problème, appelée problème maître restreint (PMR). Notre PMR est :

$$\text{minimiser } \sum_{S_s \in \tilde{S}} x_s \quad (5)$$

tel que :

$$\sum_{S_s \in \tilde{S}} C_{is} x_s \geq 1 \quad \forall i \in U \quad (6)$$

$$\sum_{S_s \in \tilde{S}} D_{(ij)s} x_s \geq 1 \quad \forall (i, j) \in U^2, i < j \quad (7)$$

$$x_s \in [0, 1] \quad \forall S_s \in \tilde{S} \quad (8)$$

où $\tilde{S} \subset S$ est une collection réduite d'ensembles et x est un vecteur de variables continues de taille $|\tilde{S}|$. Comme ce PMR est une relaxation linéaire sur un très petit sous-ensemble de variables, il est plus facile à résoudre pour un solveur de programmation linéaire (PL) que la formulation PLNE d'origine. Mais comme il manque de nombreuses variables, sa solution optimale peut ne pas être optimale pour le problème complet (dans sa version relaxée). Il faut donc prouver qu'il n'existe aucune colonne supplémentaire possible qui pourrait faire partie d'une solution de coût plus faible. Si une telle colonne existe, elle doit être ajoutée à $\tilde{S}$.

Pour déterminer quelles colonnes doivent être ajoutées pour réduire le coût de la solution, un *problème de tarification* est résolu. Ce dernier se base sur la version duale du problème.

Problème dual : Pour la relaxation linéaire du modèle PLNE, le problème dual est :

$$\text{maximiser } \sum_{i \in U} \pi_i + \sum_{i < j \in U^2} \pi'_{ij} \quad (9)$$

tel que :

$$\sum_{i \in U} \pi_i C_{is} + \sum_{i < j \in U^2} \pi'_{ij} D_{(ij)s} \leq 1 \quad \forall S_s \in S \quad (10)$$

où π est un vecteur de variables duales de taille $|U|$ liées aux contraintes de couverture (une variable pour chaque contrainte de couverture dans le PMR) et π' est un vecteur de variables duales de taille $\frac{|U| \cdot (|U| - 1)}{2}$ liées aux contraintes de 1-identifiabilité (une variable pour chaque contrainte imposant la 1-identifiabilité). Ce problème dual nous permet d'en dériver le problème de tarification.

Problème de tarification : L'algorithme du simplex utilisé pour trouver la solution optimale du PMR fournit également la solution optimale $(\pi, \pi')_{\tilde{S}}$ à son problème dual. Le coût réduit d'un ensemble $S_s \in S \setminus \tilde{S}$ est donné par :

$$1 - \sum_{i \in U} \pi_i C_{is} - \sum_{i < j \in U^2} \pi'_{ij} D_{(ij)s} \quad (11)$$

Si un ensemble S_s présente un coût réduit négatif, cela signifie que la contrainte duale associée à S_s est enfreinte et la solution trouvée $(x)_{\tilde{S}}$ peut donc être non optimale. Pour remédier à cela, S_s doit être ajouté à $\tilde{S}$. Mais il est important que $\tilde{S}$ soit aussi petit que possible. Par conséquent, il n'est pas efficace d'ajouter tous les ensembles de $S \setminus \tilde{S}$ qui présentent un coût négatif. Au lieu de cela, nous sélectionnons les 10 variables présentant les coûts réduits les plus négatifs. Ce paramètre de 10 a été choisi comme compromis entre la vitesse de convergence et le nombre de colonnes ajoutées à chaque étape.

Initialisation : Pour obtenir les valeurs duales requises pour déterminer quelles colonnes doivent être ajoutées, le PMR initial doit être faisable. Il faut donc initialiser $\tilde{S}$ avec une solution viable. Pour cela, il est initialisé avec la solution renvoyée par l'algorithme glouton 1.

Solution entière : Rappelons que la génération de colonnes, lorsqu'elle atteint son critère d'arrêt, fournit une solution optimale à la relaxation linéaire du problème. Elle peut attribuer des valeurs fractionnaires aux variables binaires et ne garantit pas que l'ensemble des colonnes candidates qui en résulte contienne la solution optimale au problème entier initial. Par conséquent, nous avons besoin d'une étape supplémentaire pour obtenir une solution entière. Pour cela nous utilisons le modèle PLNE sur la collection réduite $\tilde{S}$ obtenue à la fin de la GC. Cette approche ne garantit pas de trouver la solution optimale du problème original mais permet tout de même de retourner des solutions de bonne qualité comme montré dans la section 4.

4 Résultats expérimentaux

Dans cette section, nous évaluons nos différentes approches.¹ Nous utilisons Gurobi 11.0.2 [5] pour résoudre le modèle PLNE et le PRM de la génération de colonnes. Toutes les expériences tournent sur des processeurs AMD Epyc Genoa at CPU 2.4 GHz avec jusqu'à 766 Go de mémoire. La limite de temps est fixée à 30 minutes et chaque solveur peut utiliser jusqu'à 20 GB de mémoire.

Dataset : Pour évaluer nos modèles, nous utilisons des topologies du CAIDA ITDK (topologies IPv4 de Février 2022) [3], Rocketfuel [10] et du Internet Topology Zoo [6].

1. Le code source est disponible à <https://github.com/BurlatsAlice/PathSelection>

Nous supposons que les plus courtes routes sont basées sur le nombre de hops. Pour avoir un ensemble de routes plus réaliste, nous utilisons aussi des topologies provenant de *Repetita* [4], elles contiennent des poids IGP réalistes. Parmi elles, nous retrouvons des évolutions du *Internet Topology Zoo* et de *Rocketfuel*. Ainsi, pour certaines instances nous avons deux versions. Dans ces cas-là nous utilisons celle contenant les poids IGP. Pour les topologies qui ne sont pas connexes, nous gardons seulement la composante la plus grande pour s'assurer que l'instance peut être résolue. Au total, notre dataset contient 365 topologies, avec jusqu'à 960 nœuds et 920640 routes.

Solveur	Meilleure solution trouvée	Optimalité prouvée
PLNE	335 (91.78%)	195 (53.42%)
Glouton	55 (15.07%)	None
GC	262 (71.78%)	None

TABLE 1 – Nombre d'instances résolues pour chaque modèle.

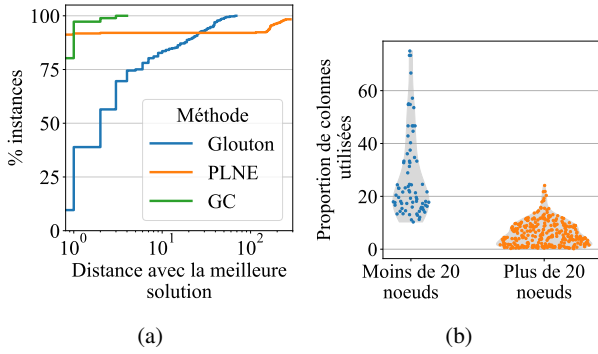


FIGURE 2 – (a) Proportion d'instances résolues par rapport à la distance sur l'objectif avec la meilleure solution trouvée. (b) Distribution de la proportions de routes sélectionnées par rapport au nombre total de routes candidates, selon le nombre de nœuds dans la topologie.

Résultats : La Table 1 montre pour chaque solveur la proportion d'instances pour lesquelles ils renvoient la meilleure solution trouvée, et la proportion d'instances pour lesquelles ils prouvent l'optimalité de leur solution. Comme le glouton et la GC ne sont pas en mesure de prouver l'optimalité de leurs solutions, la dernière valeur est None. La figure 2a montre pour chaque solveur la différence de coût entre la solution du solveur et la meilleure solution trouvée. Les résultats montrent que le modèle PLNE est prometteur : dans la Table 1 nous observons que le glouton est capable de retourner la meilleure solution trouvée pour seulement 15.07% des instances alors que cette proportion est de 91.78% pour le solveur PLNE. Pour 10,68% des instances (39 instances), le glouton renvoie une solution utilisant au moins 10 routes de plus que la meilleure solution trouvée. Ainsi, l'utilisation d'un solveur PLNE exact comme Gurobi réduit généralement le coût de surveillance.

Mais le solveur PLNE manque de mémoire pour 30 instances : parmi elles, la plus petite topologie contient 372 nœuds et 138012 routes, mais la plus grande instance qui est résolue sans problème de mémoire contient 413 nœuds et 170156 routes. Cette limitation de mémoire lui fait retourner des solutions loin d'être optimales (avec jusqu'à 283 routes supplémentaires), comme le montre la figure 2a. Dans nos instances, la plupart des routes ont des contributions similaires. Ainsi, lorsqu'une route est sélectionnée, de nombreuses autres routes voient leurs contributions devenir très limitées et deviennent donc inutiles. Le problème peut donc être résolu en n'utilisant qu'un petit sous-ensemble de variables, et donc en réduisant la mémoire consommée.

La GC permet d'exploiter cette caractéristique de nos instances : La figure 2b montre la distribution de la proportion de colonnes utilisées pour résoudre une instance avec la GC. Nous pouvons observer que, la plupart du temps, moins de 15% des routes sont ajoutées au modèle lors de la GC. Cela montre deux choses : d'une part que la GC est appropriée à ce problème, car la plupart des variables peuvent ne pas être ajoutées au problème ; d'autre part très peu de routes (parmi celles fournies par le protocole de routage) sont nécessaires pour assurer la 1-identifiabilité du réseau. Pour aucune instance le solveur GC ne rencontre de problèmes de mémoire. En ce qui concerne la qualité des solutions obtenues, le tableau 1 montre qu'en utilisant la GC, nous trouvons une solution avec un coût égal à la meilleure solution trouvée pour 71,78% des instances. La GC permet d'améliorer la solution du glouton dans 87,67% des cas. Pour 26,30% des instances, la limite de temps a été atteinte lors de la résolution du modèle PLNE final, ce qui indique que le modèle PLNE réduit reste difficile à résoudre malgré l'ensemble restreint de colonnes. Pour 1,36% des instances, la GC ne termine pas dans le temps imparti et la solution rapportée est celle utilisée pour initialiser le PMR. Au final, la GC retourne des solutions de meilleures qualités que le glouton tout en se préservant des problèmes de mémoire du modèle PLNE. Selon nous, elle représente le meilleur compromis pour résoudre ce problème.

5 Conclusion

Dans ce travail, nous nous sommes concentrés sur la sélection des routes de mesure pour garantir la 1-identifiabilité de chaque nœud dans un réseau. Nous avons introduit le problème de 1-identifiabilité par ensembles pour sélectionner l'ensemble minimal de routes de mesure. Notre comparaison d'une approche de GC avec un modèle PLNE et un algorithme glouton a révélé que la GC évite les limitations de mémoire du modèle PLNE tout en fournissant des solutions supérieures à l'algorithme glouton et des solutions de meilleure qualité que la PLNE sur les instances difficiles et de grande taille. Pour les travaux futurs, le développement d'un algorithme de « branch-and-price » en complément de la GC pourrait être une direction prometteuse pour garantir l'optimalité des solutions trouvées.

Références

- [1] Pamela Bezerra, Po-Yu Chen, Julie A. McCann, and Weiren Yu. Adaptive Monitor Placement for Near Real-time Node Failure Localisation in Wireless Sensor Networks. *ACM Transactions on Sensor Networks*, 18(1) :2 :1–2 :41, October 2021.
- [2] Auguste Burlats, Pierre Schaus, and Cristel Pelsser. An exploration of exact methods for effective network failure detection and diagnosis. In Bistra Dilkina, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 153–169, Cham, 2024. Springer Nature Switzerland.
- [3] CAIDA. The CAIDA Macroscopic Internet Topology Data Kit, February 2022.
- [4] Steven Gay, Pierre Schaus, and Stefano Vissicchio. Repetita : Repeatable experiments for performance evaluation of traffic-engineering algorithms, 2017.
- [5] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.
- [6] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *Selected Areas in Communications, IEEE Journal on*, 29(9) :1765 –1775, october 2011.
- [7] Liang Ma, Ting He, Ananthram Swami, Don Towsley, and Kin K. Leung. On optimal monitor placement for localizing node failures via network tomography. *Performance Evaluation*, 91 :16–37, September 2015.
- [8] Teresa Pepe and Marzio Puleri. Network Tomography : A novel algorithm for probing path selection. In *2015 IEEE International Conference on Communications (ICC)*, pages 5337–5341, June 2015. ISSN : 1938-1883.
- [9] Wei Ren and Wei Dong. Robust network tomography : K-Identifiability and monitor assignment. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.
- [10] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring ISP topologies with rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4) :133–145, August 2002.
- [11] Sava Stanic, Suresh Subramaniam, Gokhan Sahin, Hongsik Choi, and Hyeong-Ah Choi. Active monitoring and alarm management for fault localization in transparent all-optical networks. *IEEE Transactions on Network and Service Management*, 7(2) :118–131, June 2010. Conference Name : IEEE Transactions on Network and Service Management.

Équilibrage de ligne d'assemblage aéronautique sous contrainte de ressources avec effet d'apprentissage : une approche de programmation par contraintes

Duc-Anh Le¹, Stéphanie Roussel¹, Christophe Lecoutre²

¹ DTIS, ONERA, Université de Toulouse, France

² CRIL, Université d'Artois & CNRS, France

duc_anh.le@onera.fr, stephanie.roussel@onera.fr, christophe.lecoutre@cril.fr

Résumé

L'équilibrage des lignes d'assemblage en aéronautique est un défi majeur dans le domaine de la production. Les usines aéronautiques sont généralement conçues avec une cadence de production prédéterminée, mais le système de production nécessite un temps d'adaptation au démarrage. Ce phénomène, appelé effet d'apprentissage, correspond à l'amélioration progressive de l'efficacité par la répétition des tâches, réduisant ainsi leur durée. Cependant, la stabilité de la ligne est également un facteur clé, car toute modification du processus de production engendre un coût. Cette étude utilise la programmation par contraintes pour optimiser l'équilibrage des lignes d'assemblage, en intégrant l'effet d'apprentissage afin de gérer le compromis entre l'atteinte d'une cadence de production cible et la minimisation des ajustements apportés à la ligne d'assemblage.

Mots-clés

Équilibrage de ligne d'assemblage, ressources contraintes, effet d'apprentissage, aéronautique, programmation par contraintes.

1 Introduction

Le problème d'équilibrage des lignes d'assemblage (Assembly Line Balancing Problem – ALBP) est un défi majeur d'optimisation en production industrielle. L'objectif est de répartir équitablement les charges de travail entre les postes (les stations de travail) tout en minimisant les temps d'inactivité et en s'assurant que la cadence de production répond à la demande. Il existe différentes catégories et configurations de lignes d'assemblage, et les lignes d'assemblage aéronautiques sont généralement des systèmes droits à modèle unique, comme illustré dans la figure 1. La cadence de production d'une ligne d'assemblage est représentée par son temps-de-cycle (TdC), c'est-à-dire le temps entre la finalisation de l'assemblage de deux exemplaires consécutifs. Le TdC correspond également au temps imparti à chaque poste de travail pour réaliser les tâches qui lui sont allouées avant de transmettre l'exemplaire au poste suivant.

Les usines de fabrication d'avion sont conçues pour pro-

duire à une certaine cadence. Cependant, une fois la production lancée, il faut du temps au système pour atteindre cette cadence de production. Les opérateurs doivent acquérir de l'expérience, améliorer leurs compétences, optimiser la gestion de la chaîne d'approvisionnement et affiner le processus de production. Ainsi, la cadence de production initiale des usines est toujours inférieure à la cadence cible, et l'écart entre les deux peut être significatif. À mesure que la production progresse, les améliorations continues dans l'optimisation des flux de travail, l'intégration de l'automatisation et l'accumulation d'expérience permettent de réduire cet écart. Ce phénomène, connu sous le nom d'*effet d'apprentissage*, désigne l'amélioration progressive de l'efficacité du système de production grâce à la répétition des tâches, ce qui entraîne une réduction de leur durée [25]. La variation de la durée d'exécution d'une tâche en fonction de sa répétition est appelée *courbe d'apprentissage* [26, 15], et la période durant laquelle cette durée s'améliore continuellement jusqu'à atteindre un état stable est appelée *phase d'apprentissage*. La période nécessaire au système de production pour atteindre la cadence de production cible est appelée *phase de montée en cadence*.

Comme le temps nécessaire pour effectuer une tâche dans un environnement de fabrication diminue au fil du temps en raison de l'effet d'apprentissage, la conception de la ligne d'assemblage et du processus d'assemblage est continuellement affinée. Cependant, la stabilité de la ligne est également un facteur critique, car toute modification du processus de production entraîne des coûts. Plusieurs causes potentielles expliquent ce problème : le temps nécessaire

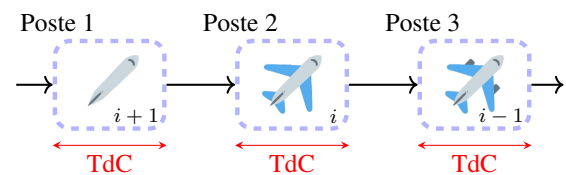


FIGURE 1 – Ligne d'assemblage droite à modèle unique, trois postes de travail. $i-1$, i et $i+1$ sont des exemplaires d'avion successifs

pour installer ou désinstaller les machines lourdes entre les postes de travail, les erreurs commises par les technicien(ne)s lorsque le processus d'assemblage change et devient moins familier, ou encore les modifications dans la gestion des stocks et de la chaîne d'approvisionnement, *etc.* Cela crée un dilemme pour les fabricants : optimiser continuellement la ligne *versus* minimiser le coût des ajustements apportés au processus de production.

L'objectif de ce travail est d'étudier l'utilisation de la Programmation par Contraintes (Constraint Programming – CP) pour traiter l'ALBP sous contraintes de ressources en tenant compte de l'effet d'apprentissage. Nous désignons ce problème par RC-ALBP/L. La décision implique un compromis entre le temps de montée en cadence – le temps nécessaire pour atteindre le taux de production cible ou le TdC – et le nombre d'ajustements apportés à la ligne – le nombre de fois où on change le TdC. Cette décision consiste en l'allocation des tâches et des ressources à chaque poste de travail, ainsi que le TdC pour chaque période de production.

Cet article présente un travail préliminaire, dans lequel nous définissons formellement RC-ALBP/L. L'encodage en CP ainsi que les expériences associées sont encore en cours de développement, et sa présentation formelle dépasse le cadre de cet article. La structure de l'article est la suivante. Après la présentation de quelques travaux connexes dans la Section 2, RC-ALBP/L est introduit formellement dans la Section 3, avec un exemple illustratif du problème. Enfin, nous concluons et discutons des perspectives pour les travaux futurs dans la Section 4.

2 Etat de l'art

L'ALBP est bien connu dans la littérature. À partir de la version simple de l'ALBP (SALBP) [16], de nombreuses extensions ont été proposées au fil des années [6, 7, 8]. Nous nous concentrons spécifiquement sur l'ALBP sous contraintes de ressources (RC-ALBP) et sur l'ALBP tenant compte de l'effet d'apprentissage.

L'ALBP sous contraintes de ressources. Ce problème a été désigné par plusieurs noms dans le passé [9, 24], mais il n'a été formellement défini qu'en 2005 dans [2]. La définition du RC-ALBP a été davantage précisée dans [12], où les auteurs ont souligné que les ressources requises par les tâches peuvent être simples ou multiples, alternatives et/ou concurrentes. La complexité du RC-ALBP a été démontrée comme étant fortement NP-difficile dans [20], même sans prendre en compte les aspects de planification. Lorsque des ressources cumulatives sont considérées, il faut non seulement allouer les tâches aux stations mais aussi décider de leur date de début. Cette variante particulière du RC-ALBP a été largement étudiée sous le nom de "Multi-Manned ALBP", introduit pour la première fois dans [13]. Plusieurs approches de résolution ont été développées pour traiter le RC-ALBP, avec ou sans prise en compte de la planification, incluant des techniques basées sur la CP [3, 10, 14, 21, 22, 23, 27].

L'ALBP avec l'effet d'apprentissage. Bien que l'ALBP ait été largement étudié avec de nombreuses extensions, peu

de travaux ont pris en compte l'effet d'apprentissage, et la plupart d'entre eux appliquent l'approche basée sur la position. Une revue des travaux menés avant 2023 prenant en compte l'effet d'apprentissage dans l'ALBP est disponible dans [1]. Une version relaxée du SALBP avec effet d'apprentissage (SALBP/L) a été abordée pour la première fois dans [11]. Les auteurs y ont démontré que l'affectation optimale des tâches, tout en minimisant le TdC d'un lot de production, est déséquilibrée en présence d'un apprentissage homogène. Ce travail a été étendu au SALBP/L dans [17]. La programmation linéaire en nombres entiers (ILP) et une approche math-heuristique, basée sur la recherche à voisinage variable et la programmation dynamique, ont été utilisées pour traiter le même problème dans [5]. Dans [18], les auteurs ont proposé deux procédures de rééquilibrage pour le SALBP/L afin de réduire le nombre de postes de travail sur la ligne. Cependant, ce travail adopte une vision relaxée de l'effet d'apprentissage lors du comptage du nombre de répétitions des tâches dans l'approche d'apprentissage basée sur la position. En effet, une tâche d'un exemplaire introduit plus tard peut être traitée avant celle d'un exemplaire introduit plus tôt, et plusieurs instances d'une même tâche peuvent être traitées simultanément pour différents exemplaires. Plus récemment, l'ALBP à modèles mixtes incorporant l'effet d'apprentissage a été abordé à l'aide d'approches ILP et math-heuristiques, étendant encore davantage les recherches dans ce domaine [4, 19].

À notre connaissance, aucun travail dans la littérature n'aborde simultanément le RCALBP en tenant compte des aspects de planification avec des durées de tâches dépendantes de l'effet d'apprentissage.

3 Description du problème

La ligne d'assemblage considérée est mono-produit et droite. De plus, en raison de la nature de la ligne d'assemblage aéronautique, nous avons introduit la notion de *zone*. Chaque poste de travail contient le même ensemble de zones, situées soit à l'intérieur, soit à l'extérieur de l'appareil. Chaque zone possède une capacité et peut être considérée comme une ressource cumulative et réutilisable pour l'exécution des tâches. En outre, pour des raisons de sécurité, certaines tâches peuvent désactiver temporairement certaines zones pendant leur exécution. Lorsqu'une zone est désactivée, toute tâche nécessitant cette zone ne peut être exécutée. Toutefois, plusieurs tâches qui désactivent la même zone peuvent être réalisées simultanément. Dans cette section, nous décrivons formellement les composants nécessaires à la définition d'une instance de RC-ALBP/L, dans le but de minimiser la durée de la phase d'apprentissage ainsi que le nombre d'ajustements de la ligne.

3.1 Entrées du problème

Une instance de RC-ALBP/L est définie par un n-uplet constitué des éléments suivants.

- Le TdC cible c_{tgt} , qui correspond au TdC que l'on souhaite atteindre.
- La valeur maximale du TdC c_{max} , c'est-à-dire le

temps maximal autorisé entre l'assemblage de deux exemplaires consécutifs.

- L'ensemble des postes de travail $\mathcal{W} = \llbracket 1, W \rrbracket$ ¹. Les postes sont ordonnés de 1 à W le long de la ligne.
- L'ensemble des périodes $\mathcal{P} = \llbracket 1, P \rrbracket$. Au début de chaque période, un nouvel exemplaire entre sur la ligne à la première station, et tous les exemplaires sur la ligne sont déplacés vers leurs stations suivantes. P peut être considéré comme le nombre maximal d'exemplaires pour atteindre la cadence cible. La durée d'une période, qui correspond au TdC de la ligne durant cette période, est une variable de décision.
- L'ensemble des tâches $\mathcal{T}$, avec pour chaque tâche $t \in \mathcal{T}$, une fonction décroissante, notée $dur_t : \mathbb{Z} \rightarrow \mathbb{N}$, qui retourne la durée de la tâche en fonction du nombre de fois qu'elle a été entièrement exécutée auparavant, ou zéro si l'entrée est négative. Notons que, dans la période numéro p , le nombre d'exécutions complètes d'une tâche affectée au poste w est $p - w$. Par exemple, la durée d'une tâche t affectée au poste 3 pendant la période 2 est $dur_t(2 - 3) = 0$, puisque le poste 3 n'est pas encore en service pendant la période 2. Pendant la période 5, sa durée est $dur_t(5 - 3) = dur_t(2)$, car elle a été exécutée deux fois (période 3 et période 4).
- Le graphe $\mathcal{G} = (\mathcal{T}, \mathcal{A})$ représente la relation de précedence entre les tâches dans $\mathcal{T}$. Un arc (t, u) indique que t doit être terminée avant le début de u .
- L'ensemble des ressources $\mathcal{R}$, où pour chaque ressource $r \in \mathcal{R}$, $capa_r$ désigne la capacité de r , et $cons_{r,t}$ désigne la quantité de ressource r consommée par la tâche $t \in \mathcal{T}$. Les ressources peuvent généralement représenter un ensemble de machines ou un ensemble de technicien(ne)s affectés à un poste de travail pour exécuter des tâches dans ce poste.
- L'ensemble des zones $\mathcal{Z}$, où, pour chaque zone $z \in \mathcal{Z}$, $capa_z$ désigne la capacité de z , $cons_{z,t}$ désigne la place de la zone z occupée par la tâche $t \in \mathcal{T}$, et $\mathcal{D}_t \subseteq \mathcal{Z}$ est l'ensemble des zones qui sont désactivées pendant l'exécution de la tâche t .
- L'horizon temporel est relatif à chaque station de travail, et est noté $\mathcal{H} = \llbracket 0, c_{max} \rrbracket$.

Hypothèses. Une instance de RC-ALBP/L est dite bien formée si les hypothèses suivantes sont vérifiées.

1. Le nombre de périodes P est supérieur au nombre de postes de travail W .
2. Le TdC ne peut pas être modifié pendant la phase de remplissage de la ligne (les W premières périodes). Sous cette hypothèse, le TdC durant ces périodes est égal au TdC maximal parmi elles.
3. Le premier TdC est supérieur au TdC cible.

Exemple 1. Nous considérons un exemple simplifié, instance de RC-ALBP/L, composé de 2 postes de travail, 2 ressources $\{r_1, r_2\}$, chacune avec une capacité de 6, et une

zone z ayant une capacité de 1. Le TdC cible est de 12 et le nombre maximal de périodes pour la montée en cadence est de 6. La valeur maximale du TdC est de 20. Il y a 6 tâches, chacune ayant une courbe d'apprentissage individuelle, c'est-à-dire des durées dépendant du nombre d'exécutions complètes de la tâche. La consommation des ressources et de la zone par chaque tâche est représentée dans la Table 1. La tâche c est la seule à désactiver la zone z pendant son exécution. Le graphe de précédence entre les tâches est présenté dans la Figure 2.

Task	Consumption			$\mathcal{D}_t$	$dur_t(n)$ avec $n =$					
	r_1	r_2	z		0	1	2	3	4	5
a	3	1	1	-	9	8	7	6	5	5
b	2	2	1	-	10	7	4	4	4	4
c	1	1	0	z	4	4	4	4	4	4
d	0	3	0	-	8	5	3	3	2	2
e	2	1	1	-	7	6	6	5	5	5
f	1	2	0	-	5	5	4	4	3	3

TABLE 1 – Exemple : l'informations sur les tâches

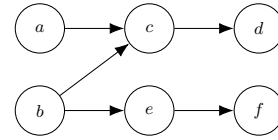


FIGURE 2 – Exemple : la relation de précédence

3.2 Affectation, solution et Pareto-optimalité

Affectation. Une *affectation* d'une instance de RC-ALBP/L est définie par un n-uplet $(wks, raw, start, cyc)$, où :

- $wks : \mathcal{T} \rightarrow \mathcal{W}$ est une fonction qui associe, à chaque tâche $t \in \mathcal{T}$, le poste $w \in \mathcal{W}$ auquel elle est affectée ;
- $raw : \mathcal{R} \times \mathcal{W} \rightarrow \mathbb{N}$ est une fonction qui associe, à chaque paire composée d'une ressource $r \in \mathcal{R}$ et d'un poste $w \in \mathcal{W}$, la quantité de r allouée au w ;
- $start : \mathcal{T} \times \mathcal{P} \rightarrow \mathbb{N}$ est une fonction qui associe, à chaque tâche $t \in \mathcal{T}$ et à chaque période $p \in \mathcal{P}$, la date de début de t relativement au poste de travail auquel elle est affectée pendant la période p ;
- $cyc : \mathcal{P} \rightarrow \mathbb{N}$ est une fonction qui associe, à chaque période $p \in \mathcal{P}$, son TdC.

Notons que nous supposons dans cette étude que l'affectation des tâches et du nombre de ressources aux stations est le même pour toutes les périodes. Cependant, l'ordonnement des tâches au sein de chaque station peut changer d'une période à l'autre.

On définit une fonction supplémentaire $end : \mathcal{T} \times \mathcal{P} \rightarrow \mathbb{N}$ qui retourne, pour chaque tâche t et chaque période p , la date de fin de t pendant p , relative à son poste de travail. La valeur de cette fonction peut être calculée comme suit :

$$end(t, p) = start(t, p) + dur_t(p - wks(t))$$

1. La notation $\llbracket 1, n \rrbracket$ désigne l'ensemble $\{1, 2, \dots, n\}$.

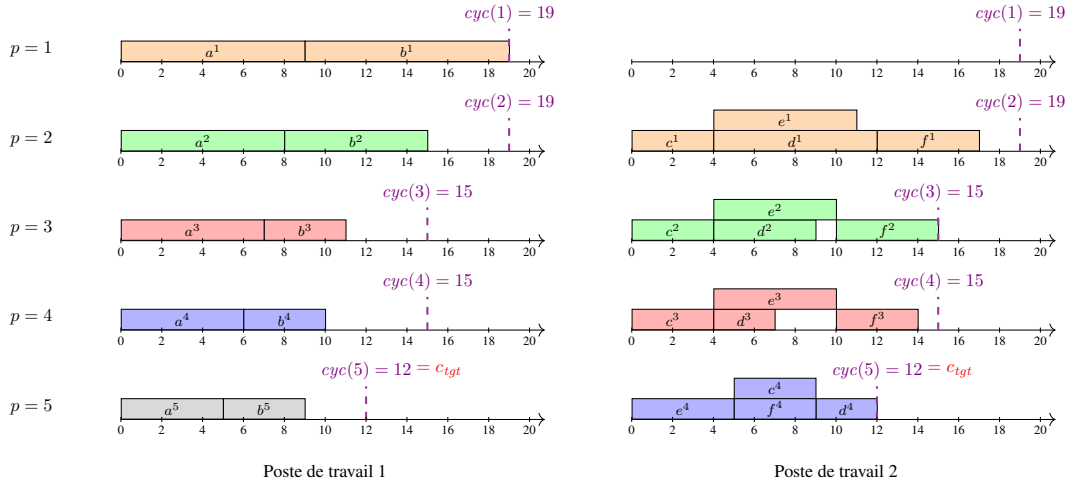


FIGURE 3 – L'affectation et l'ordonnancement des tâches pour l'exemple simplifié

On note $\mathcal{E}_{t,p} = [start(t,p), end(t,p)]$ l'intervalle de temps durant lequel la tâche t est exécutée pendant la période p .

Solution. Une affectation est une solution au problème si elle satisfait les contraintes de précédence (Eq. 1 et 2), respecte la capacité des ressources (Eq. 3 et 4) et la capacité des zones (Eq. 5), satisfait les contraintes de désactivation des zones (Eq. 6) et respecte les caractéristiques du TdC (Eq. 7 et 8).

$$\forall(t, u) \in \mathcal{A}, \quad wks(t) \leq wks(u) \quad (1)$$

$$\forall(t, u) \in \mathcal{A}, \forall p \in \mathcal{P}, \quad (wks(t) = wks(u)) \Rightarrow (end(t,p) \leq start(u,p)) \quad (2)$$

$$\forall r \in \mathcal{R}, \quad \sum_{w \in \mathcal{W}} raw(r, w) \leq capa_r \quad (3)$$

$$\forall r \in \mathcal{R}, \forall w \in \mathcal{W}, \forall p \in \mathcal{P}, \forall \tau \in \mathcal{H}, \quad \sum_{t \in \mathcal{T}: wks(t)=w, \tau \in \mathcal{E}_{t,p}} cons_{r,t} \leq raw(r, w) \quad (4)$$

$$\forall z \in \mathcal{Z}, \forall w \in \mathcal{W}, \forall p \in \mathcal{P}, \forall \tau \in \mathcal{H}, \quad \sum_{t \in \mathcal{T}: wks(t)=w, \tau \in \mathcal{E}_{t,p}} cons_{z,t} \leq capa_z \quad (5)$$

$$\forall t \in \mathcal{T}, \forall p \in \mathcal{P}, \forall \tau \in \mathcal{H}, \quad (\tau \in \mathcal{E}_{t,p}) \Rightarrow (\mathcal{O}_{u,\tau} = \emptyset) \text{ avec } \mathcal{O}_{u,\tau} = \quad (6)$$

$$\{u \in \mathcal{T} \setminus \{t\} \mid wks(u) = wks(t), \tau \in \mathcal{E}_{u,p}, \exists z \in \mathcal{D}_t : cons_{z,u} > 0\}$$

$$\forall t \in \mathcal{T}, \forall p \in \mathcal{P}, \quad end(t,p) \leq cyc(p) \quad (7)$$

$$\forall p \in \mathcal{P}, \quad c_{tgt} \leq cyc(p) \leq c_{max} \quad (8)$$

Pareto-optimalité. Une solution au problème est dite Pareto-optimale si aucune autre solution n'améliore l'un des deux critères d'optimisation suivants sans détériorer l'autre : 1) la durée de la phase d'apprentissage (Eq. 9), et 2)

le nombre d'ajustements de la ligne (Eq. 10), qui représente le critère de stabilité.

$$\sum_{p \in \mathcal{P}: c_{tgt} < cyc(p)} cyc(p) \quad (9)$$

$$\sum_{p \in \mathcal{P}: p > W} (cyc(p-1) \neq cyc(p)) \quad (10)$$

Exemple 2. Une solution dominante pour l'exemple précédent est partiellement illustrée sur la figure 3. Les tâches $\{a, b\}$ sont affectées au poste de travail 1, et les tâches $\{c, d, e, f\}$ au poste de travail 2. Dans le poste 1, l'ordre des tâches reste inchangé tout au long du processus de production. Dans le poste 2, pendant la période 1, aucune tâche n'est exécutée, car le premier exemplaire se trouve encore au poste de travail 1. Il n'y a aucun ajustement durant la période 2 en raison de l'hypothèse 2 du problème. Les ajustements de cadence sont réalisés pendant les périodes 3 et 5. En période 3, l'ordre d'exécution des tâches reste identique à celui de la période 2, mais une décision est prise pour réduire le TdC de 19 à 15, car le temps nécessaire pour ce poste est réduit durant cette période grâce à l'effet d'apprentissage. En période 5, une décision est prise pour modifier l'ordre des tâches et réduire davantage le TdC jusqu'au temps cible. La durée de la phase d'apprentissage dans cet exemple est de $(19 \times 2) + (15 \times 2) = 68$, et le nombre d'ajustements de la ligne est de 2.

4 Conclusion et travaux futurs

Dans cet article, nous avons formellement introduit une extension d'ALBP, appelée RC-ALBP/L, qui permet d'adresser la montée en cadence. La prochaine étape de ce travail consistera à implémenter le modèle proposé dans un solveur de CP, puis à mener des expérimentations à partir de données industrielles.

Remerciements

Ce travail a bénéficié du soutien de l'ANR (France 2030), dans le cadre du projet MAIA (ANR-22-EXES-0009).

Références

- [1] Z.Z. El Abidine and T. Koltai. The effect of learning on assembly line balancing : A review. *Periodica Polytechnica Social and Management Sciences*, 2023.
- [2] K. Ağpak and H. Gökçen. Assembly line balancing : Two resource constrained cases. *International Journal of Production Economics*, 96(1) :129–140, 2005.
- [3] H.M. Alakaş, M. Pınarbaşı, and M. Yüzüak. Constraint programming model for resource-constrained assembly line balancing problem. *Soft Computing*, 24 :5367–5375, 2020.
- [4] E. Alhomaidhi. Enhancing efficiency and adaptability in mixed model line balancing through the fusion of learning effects and worker prerequisites. *International Journal of Industrial Engineering Computations*, 15(2) :541–552, 2024.
- [5] Z. Bao and L. Chen. A matheuristic approach for the aircraft final assembly line balancing problem considering learning curve. In *2023 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 1389–1393. IEEE, 2023.
- [6] O. Battaïa and A. Dolgui. A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, 142(2) :259–277, 2013.
- [7] O. Battaïa and A. Dolgui. Hybridizations in line balancing problems : A comprehensive review on new trends and formulations. *International Journal of Production Economics*, 250 :108673, 2022.
- [8] N. Boysen, P. Schulze, and A. Scholl. Assembly line balancing : What happened in the last fifteen years? *European Journal of Operational Research*, 301(3) :797–814, 2022.
- [9] J. Bukchin and M. Masin. Multi-objective design of team oriented assembly systems. *European Journal of Operational Research*, 156(2) :326–352, 2004.
- [10] Z.A. Çil and D. Kizilay. Constraint programming model for multi-manned assembly line balancing problem. *Computers & Operations Research*, 124 :105069, 2020.
- [11] Y. Cohen, G. Vitner, and S.C. Sarin. Optimal allocation of work in assembly lines for lots with homogeneous learning. *European Journal of Operational Research*, 168(3) :922–931, 2006. Balancing Assembly and Transfer lines.
- [12] A. Corominas, L. Ferrer, and R. Pastor. Assembly line balancing : general resource-constrained case. *International Journal of Production Research*, 49(12) :3527–3542, 2011.
- [13] S.G. Dimitriadis. Assembly line balancing and group working : A heuristic procedure for workers' groups operating on the same product and workstation. *Computers & Operations Research*, 33(9) :2757–2774, 2006. Part Special Issue : Anniversary Focused Issue of Computers & Operations Research on Tabu Search.
- [14] E.B. Edis. Constraint programming approaches to disassembly line balancing problem with sequencing decisions. *Computers & Operations Research*, 126 :105111, 2021.
- [15] E.H. Grosse, C.H. Glock, and S. Müller. Production economics and the learning curve : A meta-analysis. *International Journal of Production Economics*, 170 :401–412, 2015.
- [16] W.B. Helgeson and D.P. Birnie. Assembly line balancing using the ranked positional weight technique. *Journal of industrial engineering*, 12(6) :394–398, 1961.
- [17] T. Koltai and N. Kalló. Analysis of the effect of learning on the throughput-time in simple assembly lines. *Computers & Industrial Engineering*, 111 :507–515, 2017.
- [18] Y. Li and T.O. Boucher. Assembly line balancing problem with task learning and dynamic task reassignment. *The International Journal of Advanced Manufacturing Technology*, 88 :3089–3097, 2017.
- [19] Y. Li, D. Liu, and I. Kucukkoc. Mixed-model assembly line balancing problem considering learning effect and uncertain demand. *Journal of Computational and Applied Mathematics*, 422 :114823, 2023.
- [20] D. Ogan and M. Azizoglu. A branch and bound method for the line balancing problem in u-shaped assembly lines with equipment requirements. *Journal of Manufacturing Systems*, 36 :46–54, 2015.
- [21] X. Pucel and S. Roussel. Constraint Programming Model for Assembly Line Balancing and Scheduling with Walking Workers and Parallel Stations. In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307, pages 23 :1–23 :21, 2024.
- [22] S. Roussel, T. Polacsek, and A. Chan. Assembly line preliminary design optimization for an aircraft. In *CP 2023 (The 29th International Conference on Principles and Practice of Constraint Programming)*, 2023.
- [23] K.E. Stecke and M. Mokhtarzadeh. Balancing collaborative human-robot assembly lines to optimise cycle time and ergonomic risk. *International Journal of Production Research*, 60(1) :25–47, 2022.
- [24] J.M. Wilson. Formulation of a problem involving assembly lines with multiple manning of work stations. *International Journal of Production Research*, 24(1) :59–63, 1986.
- [25] T.P. Wright. Factors affecting the cost of airplanes. *Journal of the aeronautical sciences*, 3(4) :122–128, 1936.

- [26] L.E. Yelle. The learning curve : Historical review and comprehensive survey. *Decision sciences*, 10(2) :302–328, 1979.
- [27] Z.A. ÇİL. An exact solution method for multi-manned disassembly line design with and/or precedence relations. *Applied Mathematical Modelling*, 99 :785–803, 2021.

Le Projet Harmoniser : un Outil Pratique à Quatre Couches pour la Composition Musicale Basé sur les Contraintes

Damien Sprockeels¹, Peter Van Roy¹

¹ Université catholique de Louvain, ICTEAM

{damien.sprockeels,peter.vanroy}@uclouvain.be

Résumé

Le projet Harmoniser réalise un outil basé sur la programmation par contraintes pour aider les compositeurs de musique. Nous définissons un modèle de composition pour la musique tonale en quatre couches. Les quatre couches sont (1) la structure globale, (2) les progressions d'accords et les modulations, (3) la disposition des accords à quatre voix, et (4) les ornements. Nous avons réalisé les couches 2 et 3 dans le solveur Gecode. Nous sommes en train de réaliser les autres couches avec une interface graphique comme plug-in pour une station de travail audio numérique.

Mots-clés

Musique tonale, Station de travail audio numérique, programmation par contraintes.

Abstract

The Harmoniser project is building a tool to aid music composers in creating musical pieces using constraint programming. We define a four-layer composition process for tonal music that is useful for a significant number of composers. The four layers are (1) global structure, (2) chord progressions and modulations, (3) voicing (four-voice chord layout), and (4) ornaments (adding local complexity). We have defined layers 2 and 3 and implemented them in the Gecode solver. We are building the remaining layers with a GUI as a plug-in for a Digital-Audio Workstation.

Keywords

Tonal music, Digital-Audio Workstation (DAW), constraint programming.

1 Introduction

La composition musicale est depuis longtemps considérée comme une application prometteuse de la programmation par contraintes (PC), car la musique tonale occidentale est hautement combinatoire et repose sur une base théorique solide ([9, 7]). La théorie musicale et la composition musicale sont deux domaines vastes, et nous ne prétendons pas les aborder de manière exhaustive. Nous avons plutôt choisi un sous-ensemble pertinent de la théorie musicale tonale et proposé un processus de composition qui, bien que non général, est utile à un nombre significatif de compositeurs.

Travaux antérieurs La programmation par contraintes est une technique populaire pour la génération ([17, 18, 4]). La génération musicale avec PC est également populaire. Elle est parfois ajoutée à certaines formes d'apprentissage ([14, 11]) pour offrir un meilleur contrôle à l'utilisateur et une meilleure structure globale. Cependant, l'apprentissage est limité par les données d'entraînement. Une approche alternative consiste à formaliser la théorie musicale et à s'appuyer uniquement sur PC ([13, 1, 3, 5, 6]). Cette alternative présente de multiples avantages. Premièrement, il n'y a aucune limitation liée aux données d'entraînement. Deuxièmement, elle garantit que les règles sont respectées dans chaque solution générée. Troisièmement, elle facilite la manipulation des solutions, offrant ainsi plus de contrôle aux compositeurs. Cependant, elle présente deux inconvénients. Premièrement, un ensemble conséquent de règles est nécessaire, et deuxièmement, la recherche de solutions nécessite souvent un temps de calcul important. Les travaux antérieurs utilisant PC pour créer des outils de composition se divisent en deux catégories. La première concerne les outils formalisant un aspect spécifique de la théorie musicale ([8, 23, 12]). La seconde concerne les outils capables de modéliser une théorie musicale ([2, 15, 20]), mais nécessitant de la programmation de la part du compositeur, ce qui limite le public d'utilisateurs. Nous proposons une approche différente qui modélise entièrement un style musical sans nécessiter de la programmation.

Le projet Harmoniser Le projet Harmoniser définit un processus de composition à quatre couches, chaque couche contenant un modèle de contraintes. Les contraintes sont tirées de traités de théorie musicale et formalisées mathématiquement (voir Sections 3 et 4). L'application de ces contraintes par le solveur décharge le compositeur d'un travail fastidieux, qui lui permet de se concentrer sur l'ajout d'idées musicales pour façonner les solutions et obtenir le résultat souhaité. Afin de réduire la complexité de ce calcul, nous suivons la décomposition du processus de composition introduite dans [21], conformément à [16], qui conclut qu'une structuration appropriée est nécessaire pour rendre le problème résoluble. Nous décomposons le processus de composition musicale en problèmes plus petits, pouvant être résolus indépendamment (voir Section 2).

Structure Cet article donne un aperçu du projet Harmoniser en présentant quelques de ses idées les plus im-

portantes. Pour une présentation plus complète, nous renvoyons le lecteur aux publications [21, 22]. La Section 2 explique d’abord le processus de composition en quatre couches. Nous présentons ensuite les principales idées des deux couches actuellement implémentées : la Couche 2 (progressions d’accords et modulations, voir la Section 3) et la Couche 3 (les voix, voir la Section 4). Pour chaque couche, le modèle formel est défini mathématiquement par des contraintes, et son implémentation pratique est réalisée dans le solveur de contraintes Gecode [10]. La Section 5 propose une brève évaluation de l’outil actuel, notamment son efficacité de calcul et son utilisation par un compositeur. Enfin, la Section 6 explique la suite du projet.

2 Processus de Composition

Il existe probablement autant de façons de composer de la musique qu’il y a de compositeurs. Dans un outil pratique, il est néanmoins important de fournir une structure prédéfinie pour guider les compositeurs. Nous avons identifié un processus itératif à quatre couches, suffisamment général pour être pertinent pour un nombre significatif de compositeurs. Nous expliquons brièvement les quatre couches.

La structure globale (Couche 1) La première couche décompose la pièce musicale complète hiérarchiquement en thèmes, où chaque thème primitif est une juxtaposition de progressions incluant des modulations. Chaque progression contient une séquence d’accords dans une tonalité. Deux progressions successives sont reliées par une modulation.

Progressions et modulations (Couche 2) La deuxième couche réalise le développement harmonique au sein de chaque progression. Elle définit une séquence de degrés d’accords pour chaque progression, ainsi que des modulations pour la transition d’une progression à la suivante. Les degrés d’accords peuvent être diatoniques (appartenant à la tonalité) ou chromatiques (n’appartenant pas). La Section 3 présente les idées principales du modèle formel de cette couche. La définition complète est donnée dans [22].

Les voix (Couche 3) La troisième couche définit les voix, c’est-à-dire les notes pour chaque accord. La disposition standard d’un accord est à quatre voix. Deux aspects sont à prendre en compte : l’harmonie, c’est-à-dire l’interaction entre les notes d’un accord donné, et la mélodie, c’est-à-dire l’interaction temporelle entre les notes d’une même voix. Certaines règles des voix influencent les états des accords, ce qui signifie que la couche 2 contient également des règles des voix. La définition complète est donnée dans [21].

Les ornements (Couche 4) La quatrième couche est celle des ornements mélodiques. Étant donné l’harmonisation d’une progression d’accords, des ornements tels que des notes de passage ou des appoggiatures sont ajoutés pour enrichir la pièce. Cela ajoute une complexité essentielle.

3 Progression d’Accords et Modulations (Couche 2)

La couche 2 définit un modèle formel de d’accords et de modulations dans un contexte tonal. Nous donnons un

	I	II	III	IV	V	VI	VII	Vda	V/II	V/III	V/IV	V/V	V/VI	V/VII	bII	6Δ
I	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
II	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
III	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IV	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
VI	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
VII	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Vda	A				1				B						C	
V/II		1								1						
V/III			1								1					
V/IV				1								1				
V/V					1								1			
V/VI						1								1		
V/VII							1								1	
bII					1											
6Δ						1										
	G								H						I	

FIGURE 1 – Matrice de transition T entre les degrés successifs d’un accord dans une tonalité, utilisée pour définir les progressions d’accords. Un emplacement vide dans la matrice correspond à la valeur zéro.

aperçu des principaux concepts du modèle (le modèle complet est présenté dans [22]). Nous définissons une *progression* comme une séquence de degrés d’accords dans une tonalité donnée et une *modulation* comme une transition entre deux progressions successives de tonalités différentes. Le modèle définit les accords possibles (degré, qualité et état) et les transitions entre eux, conformément à la théorie de la musique tonale occidentale, basée sur le concept de *tonalité*¹. Tous les concepts utilisés dans le modèle sont des concepts standards de la théorie musicale tonale, pour lesquels de nombreuses références existent².

Théorie musicale Les règles implémentées dans le modèle sont tirées de [7] et [9], garantissant ainsi la cohérence avec le modèle des voix (Couche 3) [21]. Nous utilisons le chapitre de Duha sur les modulations ainsi que les chapitres 4 (triades et accords de septième), 6 (écriture des parties), 8-11 (harmonie diatonique), 13 (accords de dominante), 14 (accords prédominants), 16 (accord 6-4), 17 (troisième et sixième degré), 19 (accord de septième de sensible), 21 (accords de dominante secondaire), 29 (accord napolitain) et 30 (accord de sixte augmentée) de Gauldin. Le modèle a été élaboré en collaboration avec deux compositeurs.

Accord Un *accord* est un ensemble de trois notes ou plus, identifié de manière unique par un triplet (r, q, s) où r est la note fondamentale (l’une des douze notes de la musique occidentale), q est la qualité (qui définit les intervalles entre les notes de l’accord) et s est l’état (défini par la note la plus grave de l’accord). Dans une tonalité donnée, chaque note possède un degré d qui définit sa fonction au sein de la tonalité, ainsi que les qualités et états possibles pour construire des accords sur cette note comme fondamentale.

Matrice de transition d’accords Le cœur du modèle de la Couche 2 est la matrice de transition T , qui définit les transitions d’accords possibles entre deux degrés d’une tonalité (voir la Figure 1). Il s’agit d’une matrice constante qui encode les transitions permises par la théorie musicale tonale. La matrice T est complétée par trois autres matrices constantes M , P et L , qui définissent respectivement les re-

1. Une tonalité est définie comme une paire composée d’une clé (l’une des douze notes C, C#, D, jusqu’à B) et d’un mode (majeur ou mineur).

2. Quelques liens : *tonalité*, *accord*, *fonction*, *dominante secondaire* et *accord de sixte augmentée*, parmi beaucoup d’autres.

lations entre un degré d'accord et les qualités, états et notes fondamentales possibles de l'accord. Ces quatre matrices encodent de manière compacte une partie conséquente de la théorie musicale, ce qui, à notre connaissance, n'a été réalisé par aucun outil de composition précédent. À ces matrices, des contraintes sont ajoutées pour modéliser des aspects spécifiques de la théorie musicale tonale qui ne sont pas capturés par les matrices. De plus, des contraintes sont ajoutées pour modéliser les modulations.

La matrice de transition T se lit comme suit : l'accord *rang* peut être suivi de l'accord *colonne* si la valeur de la matrice est égale à 1. Par exemple, l'accord III peut être suivi du VI et du V/VI, mais pas de tout autre degré. La matrice T encode les règles générales de l'harmonie tonale, tirées de [7, 9]. Chaque bloc de la matrice contraint un aspect spécifique des progressions d'accords dans un contexte tonal :

- Le bloc **A** (orange) définit la succession d'accords possible entre les accords diatoniques³.
- Le bloc **B** (bleu foncé) définit les accords de dominante secondaire⁴ qui peuvent suivre les accords diatoniques.
- Le bloc **C** (jaune) définit quels accords chromatiques peuvent suivre les accords diatoniques.
- Les blocs **D** (vert) et **E** (cyan) définissent les accords diatoniques et les autres dominantes secondaires qui peuvent suivre les dominantes secondaires. Les dominantes secondaires doivent se résoudre vers un accord basé une quinte en dessous de leur fondamentale. Elles peuvent se résoudre soit vers l'accord diatonique correspondant (par exemple, II pour V/II), soit vers un autre accord de dominante basé sur cette même note (par exemple, V/V pour V/II).
- Le bloc **F** (gris) applique les règles des accords chromatiques suivant les accords de dominante secondaire. Ce n'est pas autorisé; cette partie de la matrice est donc vide.
- Les blocs **G** (violet) et **H** (magenta) appliquent les règles des accords chromatiques. La sixte napolitaine (bII) et la sixte augmentée (6 Δ) doivent aller à V, mais avant cela ils peuvent aller à l'appoggiature du cinquième degré (Vda).
- Le bloc **I** (rouge) impose des règles de succession d'accords chromatiques. Ce n'est pas autorisé; cette partie de la matrice est donc vide.

4 Les Voix (Couche 3)

La couche 3 définit un modèle formel pour l'enchaînement à quatre voix des accords, c'est à dire la traduction d'accords (exprimés en degrés) en ensembles de notes [21]. Le modèle est composé de deux parties : un modèle de base (pour les progressions d'accords simples) et une extension (pour les progressions multiples, afin de gérer le résultat de la couche 2). Le modèle de base définit les voix d'une progression simple avec des accords diatoniques dans une tex-

ture à quatre voix, une technique courante pour représenter l'harmonie. Le modèle de base comprend des triades diatoniques et des accords de septième de dominante, y compris leurs renversements. L'extension prend en charge la juxtaposition de progressions, les accords chromatiques dans les progressions, ainsi que les septièmes d'espèce et les modulations entre progressions.

Théorie musicale De [7] nous reprenons les chapitres sur les accords de 3 et 4 sons et les renversements. De [9] nous reprenons les chapitres sur la texture à quatre voix, l'harmonie diatonique, la septième de dominante et les renversements (chapitres 5 à 11, 16 et 17).

Concepts de base La texture à quatre voix définit quatre voix distinctes, appelées basse, ténor, alto et soprano, classées de la plus grave à la plus aiguë en référence à l'amplitude des voix humaines correspondantes. Le modèle de base définit 15 règles (contraintes) et 5 préférences (contraintes optionnelles avec coûts) pour les quatre voix. Les contraintes harmoniques s'appliquent aux notes d'un même accord, tandis que les contraintes mélodiques s'appliquent aux transitions entre accords consécutifs ainsi que les voix dominantes.

5 Évaluation

Nous présentons une évaluation de l'état actuel de l'outil en deux parties. Nous abordons d'abord le temps de calcul du solveur, puis nous expliquons comment un compositeur peut utiliser l'outil. Le code source est sur github⁵.

5.1 Efficacité et nombre de solutions

Le modèle étant conçu pour offrir une liberté maximale aux compositeurs, les seules contraintes appliquées par défaut sont celles nécessaires pour garantir que les séquences d'accords générées respectent les règles de l'harmonie tonale. Dans ce cas, le nombre de solutions pour un problème donné est énorme. De plus, ces solutions sont trouvées extrêmement rapidement, même pour des problèmes de taille significative; l'efficacité n'est donc pas encore notre priorité. Par exemple, l'enchaînement d'accord pour le morceau musical présenté dans la Section 5.2 a été généré en 3 ms sur un MacBook Pro M1. Un morceau plus long, comportant 60 accords et cinq modulations, a été généré en 20 ms. Un aspect essentiel de l'utilisation de l'outil réside dans l'ajout par le compositeur d'idées musicales, sous forme de contraintes, pour guider le solveur. Nous prévoyons que cela se fasse via une interface graphique, afin que le compositeur n'ait pas besoin d'écrire de code. L'ajout d'idées musicales entraînera une augmentation du temps de calcul. Le problème passera d'un problème de satisfaction (trouver une solution valide) à un problème d'optimisation (trouver la meilleure solution), les critères d'optimisation étant fournis par le compositeur. Par exemple, la Couche 1 pourrait ajouter des contraintes globales pour appliquer la structure AABA avec les thèmes A et B, avec des contraintes de similarité entre les A, qui sont des variations d'un même thème, et une contrainte de différence pour le thème B contrasté.

3. L'appoggiature double du cinquième degré (Vda) est traitée séparément car sa fonction musicale est complètement différente.

4. Une dominante secondaire est la dominante d'un degré diatonique.

5. <https://github.com/sprockeelsd/Harmoniser>

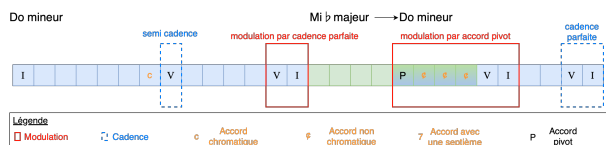


FIGURE 2 – Représentation de l'entrée au solveur.

5.2 L'utilisation par un compositeur

Nous nous mettons dans la peau d'un compositeur pour montrer comment utiliser l'outil. Nous expliquons l'exemple en termes de contraintes, mais dans un outil pratique, ces contraintes seraient fournies via une interface graphique. Pour notre exemple, nous souhaitons écrire une progression d'accords commençant en do mineur, modulant vers sa tonalité relative mi bémol majeur, puis revenant à la tonalité de do mineur, soit 28 accords. Nous souhaitons une cadence parfaite sur l'accord 12 pour terminer le premier enchaînement en do mineur et une modulation d'accord pivot de l'accord 18 à l'accord 23 pour y revenir.

En plus de ces instructions nécessaires, nous ajoutons les contraintes suivantes pour exprimer nos intentions musicales (voir la Figure 2). (1) Le premier accord doit être I. (2) Aucun accord ne peut être III ou VII, sauf pendant la modulation de l'accord pivot. (3) Seuls les accords de dominante peuvent avoir une septième. (4) Il y a une demi-cadence sur l'accord sept. (5) Il y a un accord chromatique juste avant (position six). (6) L'accord II ne doit être utilisé qu'en première inversion. (7) Il y a une cadence à la fin.

Si nous exécutons le solveur avec ces souhaits, il produit la sortie suivante. Pour la première progression en do mineur, les accords suggérés par le solveur sont *I-II-Vda-V-VI-IV-bII-V-I-V-I-IV-V-I*. Ceci est intéressant, mais selon nos goûts personnels, nous apportons quelques modifications : *I-II-Vda-V-I-IV-6Δ-V-I-V-I-Vda-V-I*. C'est également une solution acceptée par le solveur. Pour la progression de mi b majeur vers do mineur, le résultat est *I-VI-II-V-VI|I-V-VI|V-VI|I-VII|II-V-I-VI-II-V-I*, où *VI|I* est l'accord pivot démarquant la modulation (sixième degré en mi b majeur et premier degré en do mineur) et les accords suivants sont dans les deux tonalités jusqu'à la cadence en do mineur. Pour cette partie, nous n'apportons qu'une petite modification : nous modifions la cadence qui termine la modulation pour qu'elle soit interrompue. Ceci est dû au goût personnel.

Les accords finaux de cette partie sont donc *I-VI-II-V-VI|I-V-VI|V-VI|I-VII|II-V-VI-I-II-V-I*. Les états d'accords ont été omis dans la liste, mais ils sont tels qu'illustrés dans la Figure 3. Ensuite la Couche 3 génère une texture à quatre voix représentant notre morceau. La Figure 3 montre une texture à quatre voix possible. Elle peut être écoutée ici⁶. Un rythme harmonique a été donné aux accords, ainsi que quelques notes ornementales, par le compositeur. Nous prévoyons que le rythme et les notes ornementales soient fournis par l'outil dans une prochaine version.

6. <https://soundcloud.com/anonymous123-414149293/ijcai-example>



FIGURE 3 – Pièce musicale basée sur la progression d'accords générée par la Couche 2. Les voix ont été ajoutées par la Couche 3 et le rythme par le compositeur.

6 Travaux Futurs

L'outil Harmoniser complet intégrera les modèles des quatre couches dans un seul système. Pour compléter les contraintes harmoniques, nous ajouterons des contraintes rythmiques aux quatre modèles. Nous travaillons sur une application intégrant un système de gestion des solutions afin que le compositeur puisse itérer sur différentes parties de la pièce musicale, conserver les bons résultats, ajouter ou supprimer des idées musicales, combiner des solutions partielles et écouter des solutions. Nous travaillons sur une interface utilisateur graphique permettant au compositeur d'utiliser l'outil sans aucune programmation. L'application sera fournie sous forme de plug-in pour une station de travail audionumérique (DAW) afin de pouvoir être intégrée à l'atelier du musicien. Nous faisons actuellement un plug-in expérimental utilisant la plateforme JUCE [19].

L'application Harmoniser sera évaluée par des compositeurs qui donneront leur avis sur son ergonomie. Cette évaluation s'inscrit dans le cadre de notre collaboration continue avec Karim Haddad de l'IRCAM et Adrien Tsilogianis du Conservatoire royal de Bruxelles. Nous espérons que l'application finale sera utilisable par un large public de compositeurs amateurs et servira de base à de futures extensions visant à l'adapter et à l'étendre.

7 Conclusion

Cet article présente l'état d'avancement du projet Harmoniser, qui vise à développer un outil de composition basé sur la programmation par contraintes. Ce projet définit un processus de composition en quatre couches. Nous avons développé la Couche 2 (progressions d'accords et modulations) et la Couche 3 (les voix). Les travaux en cours portent sur la création des Couches 1 et 4, l'ajout de contraintes rythmiques et la création d'un plug-in avec interface graphique pour une station de travail audionumérique.

Remerciements

Les auteurs souhaitent remercier le Dr. Karim Haddad de l'IRCAM et le Pr. Adrien Tsilogianis du Conservatoire royal de Bruxelles pour leurs conseils.

Références

- [1] Torsten Anders. *Composing Music by Composing Rules : Design and Usage of a Generic Music Constraint System*. PhD thesis, Queen’s University Belfast, 2008.
- [2] Torsten Anders, Christina Anagnostopoulou, and Michael Alcorn. Strasheela : Design and Usage of a Music Composition Environment Based on the Oz Programming Model. In *Multiparadigm Programming in Mozart/Oz : Second International Conference (MOZ 2004)*, pages 277–291. Springer, 2005.
- [3] Torsten Anders and Eduardo R Miranda. A Computational Model that Generalises Schoenberg’s Guidelines for Favourable Chord Progressions. In *proceedings of the Sound and Music Computing Conference*, pages 48–52, 2009.
- [4] Alexandre Bonlarron and Jean-Charles Régin. Intertwining CP and NLP : The Generation of Unreasonably Constrained Sentences. In *Thirty-Third International Joint Conference on Artificial Intelligence {IJCAI-24}*, pages 7600–7608. International Joint Conferences on Artificial Intelligence Organization, 2024.
- [5] Grégoire Carpentier, Gérard Assayag, and Emmanuel Saint-James. Solving the Musical Orchestration Problem using Multiobjective Constrained Optimization with a Genetic Local Search Approach. *Journal of Heuristics*, 16 :681–714, 2010.
- [6] Stephen Davismoon and John Eccles. Combining Musical Constraints with Markov Transition Probabilities to Improve the Generation of Creative Musical Structures. In *European Conference on the Applications of Evolutionary Computation*, pages 361–370. Springer, 2010.
- [7] Isabelle Duha. *L’Harmonie en Liberté : de la Mémoire à l’Improvisation*. Gérard Billaudot, Armiane Imp., 2016.
- [8] Kemal Ebcioglu. An Expert System for Harmonizing Chorales in the Style of JS Bach. *The Journal of Logic Programming*, 8(1-2) :145–185, 1990.
- [9] Robert Gauldin. *Harmonic Practice in Tonal Music. Second Edition*. W. W. Norton and Company, Inc, 2004.
- [10] Gecode Team. *Gecode : Generic Constraint Development Environment*, 2019.
- [11] Luca Giuliani, Francesco Ballerini, Allegra De Filippio, and Andrea Borghesi. MusiComb : a Sample-based Approach to Music Generation Through Constraints. In *2023 IEEE 35th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 194–198, 2023.
- [12] Dorien Herremans and Kenneth Sörensen. Composing Fifth Species Counterpoint Music with a Variable Neighborhood Search Algorithm. *Expert systems with applications*, 40(16) :6427–6437, 2013.
- [13] Cheng Zhi Anna Huang and Elaine Chew. Palestrina Pal : a Grammar Checker for Music Compositions in the Style of Palestrina. In *Proceedings of the 5th Conference on Understanding and Creating Music*, 2005.
- [14] Stefan Lattner, Maarten Grachten, and Gerhard Widmer. Imposing Higher-level Structure in Polyphonic Music Generation using Convolutional Restricted Boltzmann Machines and Constraints. *Journal of Creative Music Systems*, 2 :[1]–31, 2018.
- [15] Mikael Laurson and Mika Kuuskankare. Extensible Constraint Syntax through Score Accessors. In *Journées d’Informatique Musicale*, 2005.
- [16] François Pachet and Pierre Roy. Musical Harmonization with Constraints : A Survey. *Constraints*, 6 :7–19, 2001.
- [17] François Pachet and Pierre Roy. Markov Constraints : Steerable Generation of Markov Sequences. *Constraints*, 16(2) :148–172, 2011.
- [18] Alexandre Papadopoulos, Pierre Roy, Jean-Charles Régin, and François Pachet. Generating all Possible Palindromes from n-gram Corpora. In *IJCAI 2015*, 2015.
- [19] Raw Material Software Limited. *JUCE Audio Application Development Framework*, 2025.
- [20] Örjan Sandred. PWMCMC, a Constraint-solving System for Generating Music Scores. *Computer Music Journal*, 34(2) :8–24, 2010.
- [21] Damien Sprockeels and Peter Van Roy. Expressing Musical Ideas with Constraint Programming Using a Model of Tonal Harmony. In *International Joint Conference on Artificial Intelligence*, 2024.
- [22] Damien Sprockeels and Peter Van Roy. Towards a Practical Tool for Music Composition : Using Constraint Programming to Model Chord Progressions and Modulations. In *International Joint Conference on Artificial Intelligence*, 2025.
- [23] Charlotte Truchet, Gérard Assayag, and Philippe Codognet. OMClouds, Petits Nuages de Contrainte dans OpenMusic. In *Journées d’Informatique Musicale*, 2003.

Session 3a, lundi 30 juin, 16h30-17h20 : Heuristiques

Heuristiques boîte-noire de sélection de valeur pour les problèmes d'optimisation en programmation par contraintes

Augustin Delecluse^{1,2}, Pierre Schaus¹

¹ ICTEAM, UCLouvain, Belgique

² TRAIL, Belgique

{augustin.delecluse,pierre.schaus}@uclouvain.be

5 mai 2025

Résumé

La Sélection de Valeur Impactant les Bornes est une heuristique de sélection de valeur efficace mais coûteuse, qui estime l'impact de chaque valeur d'une variable sur l'objectif via un calcul de point fixe. Nous introduisons deux améliorations à son fonctionnement : (1) restreindre le calcul du point fixe aux contraintes situées sur le plus court chemin vers l'objectif, et (2) un regard-avant inversé fixant de manière optimiste la variable objectif à sa meilleure borne. Validées sur des instances académiques et lors de la compétition XCSP³, ces approches réduisent sensiblement les coûts de calcul.

Mots-clés

Programmation par contraintes, sélection de valeur, optimisation

Abstract

Bound-Impact Value Selection is an effective but costly value selection heuristic that estimates the impact of each possible value on the objective through a fixpoint computation. We introduce two improvements to this approach : (1) restricting the fixpoint computation to constraints on the shortest path to the objective, and (2) a reversed look-ahead that optimistically sets the objective variable to its best bound. Validated on academic instances and in the XCSP³ competition, these approaches significantly reduce computation costs.

Keywords

Constraint programming, value selection, optimization

Cet article est un résumé de [7]

1 Introduction

Le paradigme de la Programmation par Contraintes (PPC) permet de résoudre des problèmes combinatoires de façon déclarative. La vision « Graal » de la PPC postule que l'utilisateur se contente de spécifier le problème, et l'ordinateur le résout [4]. Toutefois, afin de garantir une efficacité pratique, cette vision a été nuancée par le mantra historique de la PPC : *PPC = Modélisation + Recherche*. La capa-

cité à programmer des recherches spécifiques demeure cruciale pour réduire la taille de l'arbre de recherche considéré. Au fil du temps, l'intérêt porté à la programmation de la recherche a toutefois diminué grâce à l'apparition de méthodes génériques efficaces [6, 3, 9, 10].

Une procédure de recherche typique effectuée, à chaque nœud, deux choix successifs : la sélection d'une variable non encore assignée, puis le choix d'une valeur que l'on affecte à cette variable dans la branche de gauche. Malgré les nombreuses études consacrées à la conception d'heuristiques de sélection de variables basées sur le principe du *premier-échec*, peu de stratégies "boîte noire" ont été développées pour la sélection de valeur. Dans le contexte de l'optimisation, une méthode générique simple et efficace est la *Sélection de Valeur Impactant les Bornes* (SVIB) [8]. Cette heuristique de *regard-avant* évalue chaque valeur dans le domaine d'une variable : elle assigne successivement la variable à chacune de ces valeurs et calcule le point fixe induit par les contraintes. On mesure alors son impact sur l'objectif — en particulier, l'augmentation de la borne inférieure pour un problème de minimisation. La valeur entraînant l'augmentation la plus faible est sélectionnée pour la branche de gauche.¹

Bien que SVIB propose des solutions de meilleure qualité que d'autres heuristiques plus naïves (par exemple le choix systématique de la valeur minimale du domaine), le gain en guidage de la recherche peut être coûteux à obtenir. Cela est particulièrement vrai lorsque les domaines sont de grande taille ou lorsque le nombre de contraintes impliquées dans le point fixe est élevé. Une approche permettant de diminuer ce surcoût consiste à n'examiner que les valeurs extrêmes (minimum et maximum) du domaine, surtout si ce dernier est trop grand.

2 Contributions et conclusion

Notre travail introduit deux pistes pour réduire le coût de SVIB. La première, appelée *Point Fixe Restreint* (PFR), consiste à calculer le point fixe uniquement sur un ensemble restreint de contraintes. Plus précisément, on ne considère

1. Cette idée est proche du *branchement fort* utilisé dans les solveurs de Programmation Linéaire en Nombres Mixtes [1].

que les contraintes situées sur les plus courts chemins, dans le graphe de contraintes, qui relie la variable choisie à la variable objectif (chaque contrainte étant reliée aux variables de son domaine). Cette étape allégée n'est utilisée que dans l'heuristique de choix de valeur afin de sélectionner une valeur intéressante; la recherche complète prend, quant à elle, en compte l'ensemble des contraintes. Par exemple, la Figure 1 illustre ce principe pour un Problème du Voyageur de Commerce (PVC) avec un modèle de circuit en PPC. Les variables S_i ($i \in 0..n$), avec $n = 4$, représentent le successeur de chaque nœud dans le tour, D_i la distance entre un nœud et son successeur, et $DTot$ la distance totale, qui constitue l'objectif à minimiser.

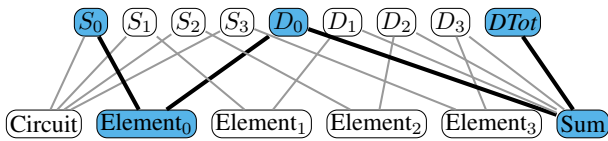


FIGURE 1 – Contraintes et variables pour un PVC à 4 nœuds. Les éléments en bleu indiquent le plus court chemin reliant la variable S_0 à l'objectif $DTot$. Seules les deux contraintes mises en évidence sont propagées lors de l'examen des valeurs de S_0 dans le cadre de PFR.

La seconde amélioration, nommée *Regard-Avant Inversé* (RAI), part de la valeur de l'objectif. Au lieu de fixer successivement la variable sélectionnée à chacune de ses valeurs (comme dans SVIB), on fixe de manière optimiste la variable objectif à sa borne inférieure (dans le cas d'une minimisation), puis on calcule le point fixe. La valeur effectivement sélectionnée pour la variable provient alors des valeurs qui subsistent après le filtrage des domaines. Si une inconsistance est détectée, on augmente la borne de l'objectif, et l'on recommence jusqu'à ce que plus aucune variable ne soit vidée de son domaine.

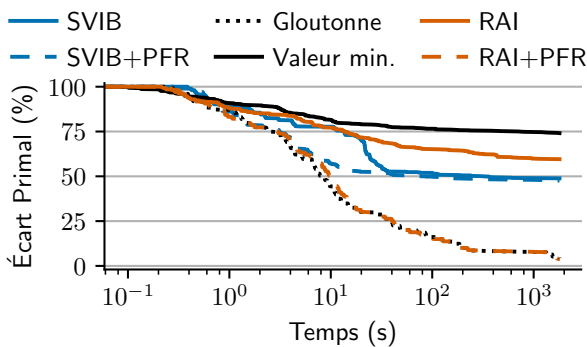


FIGURE 2 – Écart primal [5] des heuristiques au cours du temps, sur le PVC. Un écart de 100% indique qu'aucune solution n'est encore trouvée, et un écart de 0% que la meilleure solution est trouvée. L'approche RAI+PFR a le même comportement qu'une recherche gloutonne.

L'article principal [7] détaille le comportement de ces heuristiques et propose des exemples guidés. Les expérimenta-

tions, menées tant sur des problèmes académiques que sur 18 problèmes de la compétition XCSP³ 2023 [2], montrent que l'ajout de PFR dans la procédure de *regard-avant* et l'utilisation de RAI réduisent sensiblement les coûts de calcul, rendant moins pertinentes les restrictions antérieures imposées à SVIB. En particulier, elles ont le même comportement qu'une sélection gloutonne sur le PVC, comme indiqué dans la Figure 2.

Références

- [1] David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. Finding cuts in the tsp (a preliminary report), 1995.
- [2] Gilles Audemard, Christophe Lecoutre, and Emmanuel Lonca. Proceedings of the 2023 xcsp3 competition. *arXiv preprint arXiv :2312.05877*, 2023.
- [3] Gilles Audemard, Christophe Lecoutre, and Charles Prud'Homme. Guiding backtrack search by tracking variables during constraint propagation. In *International Conference on Principles and Practice of Constraint Programming*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.
- [4] Roman Barták. Constraint programming : In pursuit of the holy grail. In *Proceedings of the Week of Doctoral Students (WDS99)*, volume 4, pages 555–564. MatFyzPress Prague, 1999.
- [5] Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6) :611–614, 2013.
- [6] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *ECAI*, volume 16, page 146, 2004.
- [7] Augustin Delecluse and Pierre Schaus. Black-box value heuristics for solving optimization problems with constraint programming (short paper). In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2024.
- [8] Jean-Guillaume Fages and Charles Prud'Homme. Making the first solution good! In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1073–1077. IEEE, 2017.
- [9] Steven Gay, Renaud Hartert, Christophe Lecoutre, and Pierre Schaus. Conflict ordering search for scheduling problems. In *Principles and Practice of Constraint Programming : 21st International Conference, CP 2015, Cork, Ireland, August 31–September 4, 2015, Proceedings 21*, pages 140–148. Springer, 2015.
- [10] Christophe Lecoutre, Lakhdar Saïs, Sébastien Tabary, and Vincent Vidal. Reasoning from last conflict (s) in constraint programming. *Artificial Intelligence*, 173(18) :1592–1614, 2009.

Apprentissage et calibrage d'une heuristique générique de choix de valeur dans un solveur de programmation par contraintes.

Tom Marty^{1,2,3}, Léo Boisvert^{1,3}, Tristan François², Pierre Tessier², Louis Gautier²,
Louis-Martin Rousseau¹, Quentin Cappart^{1,3}

¹ Polytechnique Montréal, Montreal, Canada

² École Polytechnique, Palaiseau, France

³ MILA (Institut québécois d'intelligence artificielle), Montreal, Canada

23 avril 2025

Résumé

Ce papier est un résumé de l'article "Learning and Fine-Tuning a Generic Value-Selection Heuristic Inside a Constraint Programming Solver", publié dans la revue "Constraints" en novembre 2024. Notre contribution est une méthode pour apprendre des heuristiques de sélection de valeurs dans un solveur de programmation par contraintes (CP). Pour cela, nous utilisons (1) Une procédure d'apprentissage basée sur des redémarrage avec de l'apprentissage par renforcement, (2) Une fonction de récompense capable d'attribuer des valeurs intermédiaires non nulles basées sur la propagation, et (3) une architecture neuronale basée sur un réseau de neurones à graphe hétérogène.

Mots-clés

GNNs, Apprentissage par renforcement

1 Introduction et Contexte

Dans cet article, nous présentons une méthode pour apprendre des heuristiques de sélection de valeurs dans un solveur de programmation par contraintes (CP). Cette recherche est motivée par un constat : si la programmation par contraintes est une approche efficace pour résoudre des problèmes combinatoires, son efficacité dépend fortement des heuristiques de branchement utilisées.

Les heuristiques actuelles, comme la recherche basée sur l'impact [12] ou l'activité [10], présentent des limitations : elles nécessitent une initialisation coûteuse en calcul et offrent de faibles performances au début de la recherche. De plus, le développement d'heuristiques spécifiques à un problème exige une expertise approfondie, souvent inaccessible pour les utilisateurs sans formation avancée en programmation par contraintes. Cette observation a motivé de nombreux travaux visant à utiliser l'apprentissage automatique pour concevoir automatiquement des heuristiques efficaces sans intervention experte [2].

Notre contribution principale est une procédure d'apprentissage générique qui permet d'obtenir une heuristique de sélection de valeurs directement à partir d'un modèle CP en entrée. Notre approche combine trois éléments clés :

1. Une procédure d'apprentissage basée sur des redémarrages pour entraîner un agent d'apprentissage par renforcement [15] à l'intérieur d'un solveur CP.
2. Une fonction de récompense capable d'attribuer des valeurs intermédiaires non nulles basées sur la propagation effectuée pendant la recherche.
3. Une architecture neuronale basée sur une représentation de graphe tripartite et un réseau de neurones à graphe hétérogène, inspirée des travaux sur MIPs [8].

2 Contributions Techniques

2.1 Apprentissage basé sur les redémarrages

Contrairement aux approches précédentes [5] qui définissent un épisode d'apprentissage comme un processus de résolution complet utilisant le DFS, nous proposons une approche basée sur les redémarrages. Dans notre méthode, un épisode est défini comme une seule *plongée* dans l'arbre de recherche, sans retour arrière. L'épisode s'arrête lorsqu'une solution complète est trouvée ou lorsqu'un échec résulte de la dernière décision de branchement.

Cette approche présente des avantages significatifs. Premièrement, elle évite les difficultés liées à l'utilisation d'un algorithme de retour arrière dans un cadre de processus de décision markovien, problème souligné précédemment [13]. Deuxièmement, elle permet de focaliser l'apprentissage sur l'objectif de trouver rapidement de bonnes solutions, plutôt que de prouver l'optimalité (qui nécessite souvent l'exploration d'un arbre de recherche exponentiellement grand). S'inspirant des travaux récents [4, 13], notre méthode exécute chaque épisode à l'intérieur du solveur pendant l'entraînement, ce qui nous permet d'exploiter les informations mises à jour lors de la propagation pour le branchement.

2.2 Récompense basée sur la propagation

Nous avons développé un nouveau schéma de récompense basé sur la réduction du domaine de la variable choisie. Cette réduction se produit au moment de la propagation des contraintes après avoir sélectionné une valeur.

Notre fonction de récompense comprend deux composantes :

1. Une récompense intermédiaire recueillie aux nœuds de

branchement (mentionné plus haut).

2. Une récompense terminale obtenue en fin d'épisode.

Pour un problème de minimisation, la récompense intermédiaire suit deux principes : toute réduction de domaine des valeurs les plus élevées est récompensée et toute réduction des plus petites est pénalisée. Cette approche guide l'agent vers des situations où le coût minimum peut être éventuellement obtenu tout en éliminant les solutions coûteuses.

Ce mécanisme de récompense intermédiaire est particulièrement important car il permet d'atténuer le problème bien connu des récompenses clairsemées (*sparse rewards*), qui complique généralement l'apprentissage par renforcement.

2.3 Architecture de GNN hétérogène

Notre architecture neuronale est conçue pour effectuer une sélection de valeur. Elle comprend quatre étapes :

1. **Encodeur de modèle CP** : Nous représentons le modèle CP comme un graphe tripartite où les variables, les contraintes et les valeurs correspondent à des types de nœuds spécifiques et des arêtes représentant leurs relations. Cette approche s'inspire des travaux sur les MIPs [8], adaptée à la programmation par contraintes, qui implique des variables non binaires et des contraintes combinatoires. Notre encodage se distingue d'approches récentes [3, 16], offrant un meilleur équilibre entre expressivité et efficacité computationnelle.
2. **Encodeur de réseau neuronal à graphe** : Nous utilisons une architecture hétérogène où une convolution est effectuée pour chaque type de nœud. Ceci permet de propager l'information dans le graphe tout en préservant les caractéristiques de chaque type de nœud.
3. **Décodeur de réseau neuronal** : Cette étape prédit une valeur Q pour chaque valeur disponible pour la variable sélectionnée, en utilisant un algorithme d'apprentissage par renforcement profond (*deep Q-learning*) [11].
4. **Politique d'action** : Cette politique définit comment exploiter les valeurs Q approximées, équilibrant exploration et exploitation pendant la phase d'apprentissage.

3 Résultats Expérimentaux

Nous avons évalué notre approche sur quatre problèmes d'optimisation combinatoire : coloration de graphes, ensemble indépendant maximum (MIS), coupe maximum (MAXCUT), couverture de sommets minimum (MVC).

Les expériences ont été menées sur des instances de différentes tailles : petites (20-30 nœuds), moyennes (40-50 nœuds) et grandes (80-100 nœuds). L'entraînement a été effectué sur des graphes générés aléatoirement selon le modèle de Barabási-Albert [1], connu pour générer des graphes qui imitent les organisations humaines et naturelles. Nos résultats montrent que notre heuristique apprise :

- Trouve des solutions de qualité supérieure étant donné un budget de nœuds.
- Trouve la solution optimale en explorant moins de nœuds que les méthodes de référence [10, 12].
- Apprend une stratégie de branchement donnant des solutions de haute qualité, après une seule plongée.

Pour l'évaluation, nous avons utilisé des profils de performance [7] qui montrent que notre méthode surpasse les autres approches pour trouver rapidement de bonnes solutions. Par exemple, pour le problème de MAXCUT avec 50 nœuds, une seule plongée avec notre approche produit une solution avec un écart d'optimalité de 0.16, alors qu'une recherche en profondeur avec une sélection de valeur aléatoire nécessite 19 secondes et 53 000 nœuds explorés pour trouver une solution avec un écart similaire.

Les expériences montrent également que l'intégration de notre heuristique apprise dans une recherche de type ILDS [9] améliore considérablement les résultats, permettant d'obtenir des solutions très proches de l'optimalité. Nous observons finalement que le calibrage (*fine-tuning*) d'un modèle avec une classe de problèmes différente peut accélérer considérablement le processus d'apprentissage [14], comme démontré sur le problème de coupe maximum avec 80 nœuds.

4 Défis et Perspectives

Malgré les résultats prometteurs, plusieurs défis persistent.

1. **Mise à l'échelle de la représentation** : La taille des graphes augmente significativement avec la taille du problème et le nombre de nœuds dans l'arbre de recherche croît exponentiellement. Des mécanismes de compression pourraient aider à atténuer cette difficulté.
2. **Problèmes fortement contraints** : Notre approche excelle avec des solutions facilement réalisables, mais perd en efficacité sur des problèmes fortement contraints où les récompenses sont plus rares.
3. **Heuristique combinée variable/valeur** : Une direction de recherche intéressante serait d'adapter notre architecture pour apprendre simultanément une heuristique de sélection de variables et de valeurs.
4. **Preuve d'optimalité** : Notre objectif actuel est de trouver rapidement une bonne solution, mais accélérer la preuve d'optimalité reste un défi ouvert. La redéfinition de la fonction de récompense ou de l'épisode pourrait aider à aborder ce problème.
5. **Architecture de GNN** : Les GNNs basés sur le passage de messages ont des limitations intrinsèques concernant leur capacité de représentation [6]. L'exploration d'architectures alternatives est une voie prometteuse.

5 Conclusion

Nous avons proposé une méthode générique basée sur l'apprentissage par renforcement pour obtenir une heuristique de sélection de valeurs. Notre approche combine un entraînement basé sur les redémarrages, un signal de récompense non clairsemé, et une architecture de GNN hétérogène.

Les expériences sur quatre problèmes démontrent que notre méthode peut trouver de meilleures solutions en visitant significativement moins de nœuds que d'autres méthodes de référence. Nous avons également identifié plusieurs défis qui constituent des directions pour nos travaux futurs.

Références

- [1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1) :47–97, 2002.
- [2] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization : a methodological tour d’horizon. *European Journal of Operational Research*, 290(2) :405–421, 2021.
- [3] Léo Boisvert, Hélène Verhaeghe, and Quentin Cappart. Towards a generic representation of combinatorial problems for learning-based approaches. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 99–108. Springer, 2024.
- [4] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabelle Prémont-Schwarz, and Andre A. Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3677–3687, 2021.
- [5] Félix Chalumeau, Ilan Coulon, Quentin Cappart, and Louis-Martin Rousseau. Seapearl : A constraint programming solver guided by reinforcement learning. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 392–409. Springer, 2021.
- [6] Ziang Chen, Jialin Liu, Xiaohan Chen, Xinshang Wang, and Wotao Yin. Rethinking the capacity of graph neural networks for branching strategy. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
- [7] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91 :201–213, 2002.
- [8] Maxime Gasse, Didier Chetelat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 15580–15592, 2019.
- [9] William D Harvey and Matthew L Ginsberg. Limited discrepancy search. In *IJCAI*, volume 95, pages 607–615, 1995.
- [10] Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 228–243, 2012.
- [11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540) :529–533, 2015.
- [12] Philippe Refalo. Impact-based search strategies for constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 557–571. Springer, 2004.
- [13] Wen Song, Zhiguang Cao, Jie Zhang, Chi Xu, and Andrew Lim. Learning variable ordering heuristics for solving constraint satisfaction problems. *Engineering Applications of Artificial Intelligence*, 109 :104603, 2022.
- [14] Yifei Sun, Qi Zhu, Yang Yang, Chunping Wang, Tianyu Fan, Jiajun Zhu, and Lei Chen. Fine-tuning graph neural networks by preserving graph generative patterns. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8) :9053–9061, 2024.
- [15] Richard S Sutton and Andrew G Barto. Reinforcement learning : an introduction. a bradford book. *Cambridge, MA, USA*, 262039249, 2018.
- [16] Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One model, any csp : graph neural networks as fast global search heuristics for constraint satisfaction. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI ’23*, 2023.

Search is not Dead!

Christophe lecoutre

University of Artois & CNRS, CRIL

lecoute@crl.fr

Résumé

La tendance actuelle de la communauté de programmation par contraintes (PPC) à utiliser largement la technologie SAT (c'est-à-dire la traduction des contraintes et/ou le raisonnement à partir des clauses) est intéressante (et s'est avérée efficace dans bon nombre de situations), mais elle éclipse quelque peu l'intérêt de la PPC. Si l'on considère les résultats des dernières compétitions XCSP3 (2022, 2023, 2024; par exemple, voir www.cril.fr/XCSP24), on peut observer qu'un solveur PPC "pur" comme ACE peut être comparativement très efficace pour trouver des bornes de bonne qualité. En fait, notre position est que "search is not dead", une référence à l'exposé invité à la conférence CP'13 par Peter Stuckey qui affirmait que "search is dead, long live proof". Il nous semble que se concentrer uniquement sur la preuve dès le début (du processus de résolution) n'est pas nécessairement la bonne approche, en particulier avec les nouveaux progrès réalisés pour conduire la recherche de solutions (notamment, le solution-based phase saving, et les trois heuristiques complémentaires actuelles basées sur les conflits).

Mots-clés

recherche arborescente, heuristiques, contraintes

Abstract

The current trend in the CP (Constraint Programming) community to widely use SAT technology (i.e., translating constraints and/or reasoning from clauses) is interesting (and has been shown to be effective in a number of situations) but somewhat detracts from the value of CP. Considering the results of the last (2022, 2023, 2024) XCSP3 competitions (e.g., see www.cril.fr/XCSP24), one can observe that a pure CP solver like ACE can be comparatively efficient for finding good quality bounds. Actually, our position is that "search is reborn", a reference to the invited talk at the conference CP'13 by Peter Stuckey who claimed that "search is dead, long live proof". It seems to us that focusing only on proof from the start (of the solving process) is not necessarily the right approach, especially with the new advances made in search (notably, solution-based phase saving, and the three current complementary state-of-the-art conflict-based heuristics).

Keywords

tree search, heuristics, constraints

1 Introduction

La programmation par contraintes (PPC) [18, 12, 11] est une technologie utile pour modéliser et résoudre des problèmes combinatoires sous contraintes. D'une part, on peut utiliser une bibliothèque comme PyCSP³ [16] pour modéliser les problèmes qui se posent dans divers domaines d'application (par exemple, l'ordonnancement, la planification, la cryptographie, la bio-informatique, la chimie organique, etc.) Les instances de problèmes peuvent alors être directement générées à partir de modèles et de données particulières. D'autre part, pour résoudre les instances (notamment représentées au format XCSP³ [6, 7]), on peut utiliser un solveur de contraintes comme ACE, dont il est question dans le présent document. ACE est un solveur de contraintes open-source, développé en Java, qui se focalise sur les variables entières (incluant les variables booléennes ou 0/1), différentes forme de contraintes table, les contraintes globales les plus importantes, les heuristiques de recherche état-de-l'art et l'optimisation (mono-critère).

2 Vers une recherche robuste en PPC

ACE est un solveur complet, qui effectue une exploration en profondeur de l'espace de recherche, avec retours-arrières. À chaque étape (nœud de l'arbre de recherche), une décision est prise (une affectation de variable ou une réfutation de valeur) et un processus de filtrage est exécuté (mécanisme de propagation de contraintes). Pour résoudre le problème de la forte variabilité des distributions de temps d'exécution [9], la recherche est relancée régulièrement, en suivant une progression géométrique. L'ordre dans lequel les variables sont choisies au cours de la construction de la branche courante de l'arbre de recherche est déterminé par une heuristique de choix de variables; une heuristique générique classique est `dom/wdeg` [5], combinée à un mécanisme simulant une certaine forme de sauts en arrière intelligents [15]. L'ordre dans lequel les valeurs sont choisies lors de l'affectation des variables est déterminé par une heuristique de choix de valeurs; pour l'optimisation, il est fortement recommandé d'utiliser d'abord, si disponible, la valeur de la dernière solution trouvée [20, 8].

Pour l'optimisation (instances du problème COP, Constraint Optimization Problem), on utilise une stratégie basée sur la mise à jour de la borne chaque fois qu'une solution est trouvée; il s'agit d'une sorte de stratégie de descente (apparentée au Branch and Bound), dont le

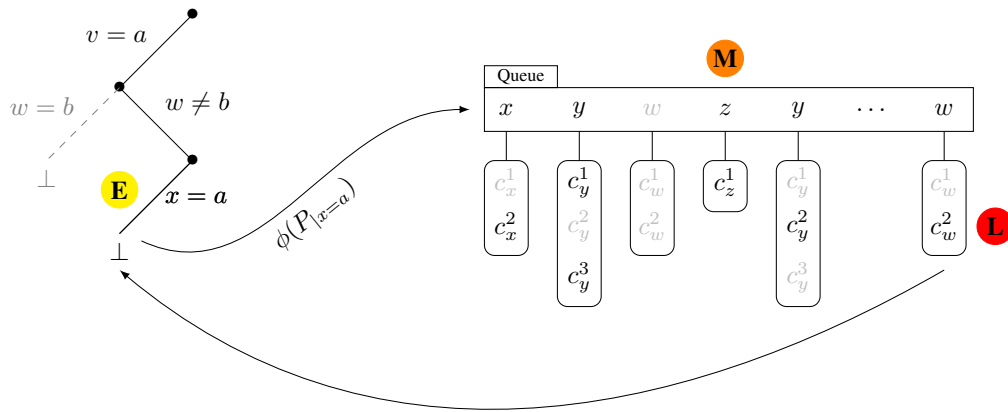


FIGURE 1 – Illustration de trois moments clés de la collecte d’informations concernant les conflits : ils correspondent au traitement précoce (Early), intermédiaire (Midway) et tardif (Late) des conflits.

principe est équivalent (dans l’hypothèse d’un problème de minimisation) à l’ajout d’une contrainte objective spéciale $\text{obj} < \infty$ au réseau de contraintes (bien qu’elle soit initialement trivialement satisfaite), et à mettre à jour la limite de cette contrainte chaque fois qu’une nouvelle solution est trouvée. Cela signifie qu’à chaque fois qu’une solution S est trouvée avec un coût $B = \text{obj}(S)$, la contrainte objective devient $\text{obj} < B$. Par conséquent, cette stratégie de descente progressive fournit une séquence de solutions de plus en plus intéressantes, jusqu’à prouver l’optimalité de la dernière solution trouvée.

Au moment de la rédaction de ce document, nous pensons que ACE est un solveur compétitif en raison des ingrédients suivants :

- redémarrage fréquent de la recherche
- enregistrement des nogoods de la branche courante au moment du redémarrage [14]
- pondération de contraintes pour la sélection des variables [5, 10, 21, 17, 4]
- raisonnement à partir du dernier conflit [15]
- solution(-based) phase saving [20, 8]

Fait important, dans [4], nous avons montré que trois manières d’exploiter les conflits pour guider la recherche présentent des comportements relativement complémentaires. Cela peut s’expliquer par le fait que les informations sont extraites à différents moments : au tout début du processus conduisant à un conflit (c’est-à-dire au moment de la décision), pendant la propagation de contraintes ou au moment où le dernier propagateur (algorithme de filtrage) est sollicité. On peut alors parler d’approches telles que le traitement opérationnel des conflits est *early* (E), *midway* (M) ou *late* (L). Ceci est illustré par la figure 1 où une nouvelle décision $x = a$ est prise, lors de la résolution d’un réseau de contraintes P , dans la continuité de deux décisions prises précédemment $v = a$ et $w \neq b$. Dans notre scénario, l’exécution de la propagation de contraintes ϕ sur (l’état actuel de) P après avoir assigné la valeur a à x , c’est-à-dire $\phi(P|x=a)$, révèle une nouvelle situation conflictuelle (dé-

signée par $\perp$). Le traitement précoce de ce nouveau conflit consiste à considérer la variable x impliquée dans la décision comme cause principale. C’est le principe de l’heuristique *frba/dom* [17]. Le traitement intermédiaire de ce conflit consiste à considérer que toutes les variables ayant joué un rôle (c’est-à-dire ayant été choisies) au cours de la propagation ont contribué à l’échec. C’est le principe qui sous-tend l’heuristique *pick/dom* [4]. Le traitement tardif de ce conflit consiste à considérer la dernière contrainte (ici, c_w^2) provoquant l’échec (c’est-à-dire supprimant la dernière valeur d’un domaine) comme l’objet d’intérêt. C’est le principe de la pondération des contraintes, comme dans *wdeg/dom* [5, 10, 21].

L’intérêt (la complémentarité) de ces trois heuristiques est démontré lors de la compétition XCSP³ 2024 (voir [3] et www.cril.fr/XCSP24), à laquelle deux versions de ACE [13] ont participé :

- ACE, avec son comportement par défaut (une seule heuristique de choix de variables est utilisée : *wdeg^{cacd}* [21] qui est un raffinement de *dom/wdeg*), et la première valeur (i.e., plus petite) est systématiquement choisie,
- ACE-mix, qui exploite les trois principales heuristiques de choix de variables mentionnées ci-dessus (ainsi que *wdeg^{cacd}*) et trois heuristiques de choix de valeurs (sélection de la valeur la plus petite, la plus grande, ou aléatoirement dans le domaine de la variable sélectionnée).

Il est clair que ACE-mix bénéficie de la complémentarité de ces heuristiques puisque cette version obtient de bien meilleurs scores que ACE dans les différents tracks de la compétition. Sans surprise, cela montre que la recherche devient (beaucoup) plus robuste lorsque l’on utilise des heuristiques complémentaires (ici, basées sur les conflits).

En fait, nous pensons que la tendance actuelle de la communauté PPC à utiliser largement la technologie SAT (c’est-à-dire traduire les contraintes en clauses et/ou raisonner à partir de clauses) est intéressante (et s’est avérée efficace dans

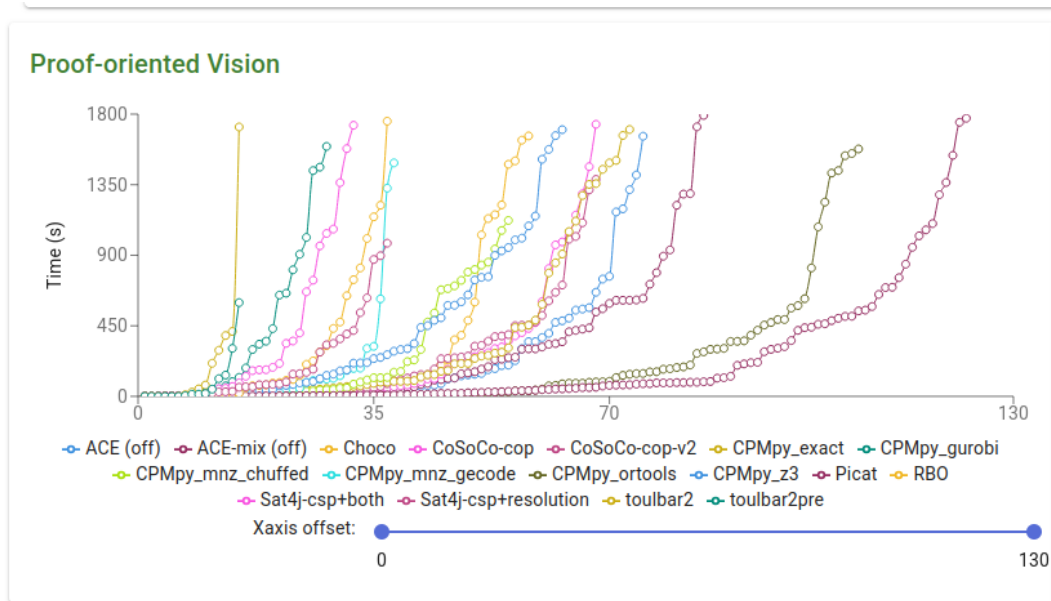


FIGURE 2 – Pour le track COP de la compétition XCSP³ 2024, avec une vision orientée preuve, Picat surpasse CPMpy_ortools, qui lui-même surpasse ACE-mix.

bon nombre de situations), mais qu’elle éclipse quelque peu la valeur de la PPC. En ce qui concerne les problèmes d’optimisation (main track COP), sur le site des résultats de la compétition XCSP³ 2024 (voir www.cril.fr/XCSP24), deux cactus plots sont visibles :

- un cactus plot, illustré à la figure 2, qui donne une vue “orientée preuve” des résultats (c’est à dire que seule les preuves d’optimalité comptent) : les approches basées sur SAT (et/ou sur la relaxation) sont très efficaces pour prouver l’optimalité ;
- un cactus plot, illustré à la figure 3, qui donne une vue “orientée recherche” (c’est à dire que les preuves d’optimalité sont ignorées) : les approches basée principalement sur la PPC sont très efficaces pour trouver des bornes de bonne qualité.

3 Search, Reborn !

Même des versions moins robustes de ACE, comme celles soumises en 2022 (voir [1] et www.cril.fr/XCSP22) et 2023 (voir [2] et www.cril.fr/XCSP23), étaient déjà très compétitives pour le main track COP. La vision orienté recherche montre déjà comparativement les très bonnes performances de ACE. En fait, notre position est que *search is not dead* (ou *search is reborn*), en référence à la conférence invitée [19] à CP’13 par Peter Stuckey qui affirmait que “search is dead, long live proof”. Comme la résolution d’un problème d’optimisation passe par trois étapes :

- la *phase d’accrochage* pendant laquelle une première solution doit être trouvée,
- la *phase de descente* pendant la quelle une solution optimale doit être trouvée, après un cheminement possiblement très long depuis la première solution,

- la *phase de preuve* pendant laquelle l’optimalité doit être prouvée,

il nous semble que se focaliser uniquement sur la preuve dès le départ n’est pas nécessairement la bonne approche, surtout avec les nouvelles avancées réalisées pour conduire la recherche (solution-based saving [20, 8] et les trois heuristiques complémentaires basées sur les conflits). Pour la phase de descente, la recherche telle qu’elle est effectuée par un solveur PPC classique peut, sous réserve d’utiliser des mécanismes de recherche complémentaires, devenir extrêmement compétitive (comme le montre ACE-mix).

Remerciements

Ce travail a bénéficié du soutien de l’ANR (France 2030), dans le cadre du projet MAIA (ANR-22-EXES-0009)

Références

- [1] G. Audemard, C. Lecoutre, and E. Lonca. Proceedings of the 2022 XCSP3 competition. Technical Report arXiv :2209.00917, CoRR, 2022. <https://arxiv.org/abs/2209.00917>.
- [2] G. Audemard, C. Lecoutre, and E. Lonca. Proceedings of the 2023 XCSP3 competition. Technical Report arXiv :2312.05877, CoRR, 2023. <https://arxiv.org/abs/2312.05877>.
- [3] G. Audemard, C. Lecoutre, and E. Lonca. Proceedings of the 2024 XCSP3 competition. Technical Report arXiv :2412.00117, CoRR, 2024. <https://arxiv.org/abs/2412.00117>.
- [4] G. Audemard, C. Lecoutre, and C. Prud’Homme. Guiding backtrack search by tracking variables during

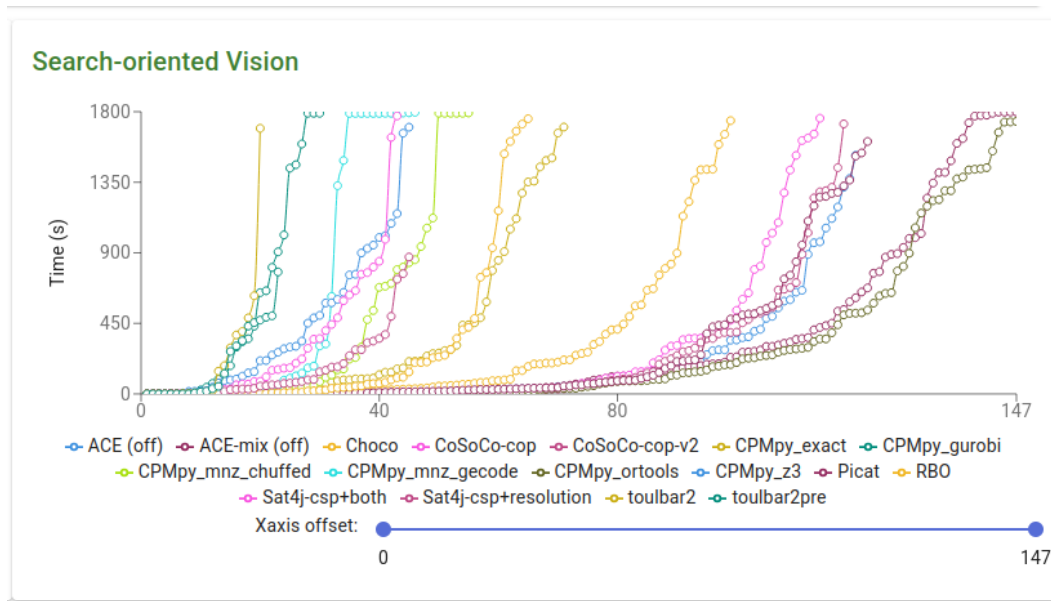


FIGURE 3 – Pour le track COP de la compétition XCSP³ 2024, avec une vision orientée recherche, ACE-mix et CPMpy_ortools sont assez proches et surpassent Picat.

- constraint propagation. In *Proceedings of CP'23*, pages 9 :1–9–7, 2023.
- [5] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.
- [6] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP3 : an integrated format for benchmarking combinatorial constrained problems. Technical Report arXiv :1611.03398, CoRR, 2016. <https://arxiv.org/abs/1611.03398>.
- [7] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP3-core : A format for representing constraint satisfaction/optimization problems. Technical Report arXiv :2009.00514, CoRR, 2020. <https://arxiv.org/abs/2009.00514>.
- [8] E. Demirovic, G. Chu, and P. Stuckey. Solution-based phase saving for CP : A value-selection heuristic to simulate local search behavior in complete solvers. In *Proceedings of CP'18*, pages 99–108, 2018.
- [9] C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24 :67–100, 2000.
- [10] D. Habet and C. Terrioux. Conflict history based search for constraint satisfaction problem. In *Proceedings of SAC'19*, pages 1117–1122, 2019.
- [11] D. E. Knuth. *Art of Computer Programming, Volume 4, Fascicle 7 : Constraint Satisfaction*. Addison Wesley, March 2025.
- [12] C. Lecoutre. *Constraint Networks : techniques and algorithms*. International Scientific and Technical Encyclopedia (ISTE Ltd) - John Wiley Inc., June 2009. 592 pages. ISBN : 9781848211063.
- [13] C. Lecoutre. ACE, a generic constraint solver. Technical Report arXiv :2302.05405, CoRR, 2023. <https://arxiv.org/abs/2302.05405>.
- [14] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Recording and minimizing nogoods from restarts. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 1 :147–167, 2007.
- [15] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173(18) :1592–1614, 2009.
- [16] C. Lecoutre and N. Szczepanski. PyCSP³ : Modeling combinatorial constrained problems in Python. Technical Report arXiv :2009.00326, CoRR, 2020. <https://arxiv.org/abs/2009.00326>.
- [17] Hongbo Li, Minghao Yin, and Zhanshan Li. Failure based variable ordering heuristics for solving CSPs. In *Proceedings of CP'21*, pages 9 :1–9 :10, 2021.
- [18] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.
- [19] P. Stuckey. Those who cannot remember the past are condemned to repeat it. In *CP'13*, pages 5–6, 2013. www.youtube.com/watch?v=IxiCHRFNng0.
- [20] J. Vion and S. Piechowiak. Une simple heuristique pour rapprocher DFS et LNS pour les COP. In *Proceedings of JFPC'17*, pages 39–45, 2017.
- [21] H. Watez, C. Lecoutre, A. Paparrizou, and S. Tabary. Refining constraint weighting. In *Proceedings of ICTAI'19*, pages 71–77, 2019.

Session 3b, lundi 30 juin, 17h20-18h00 : Applications

Analyse de l'autostabilisation de l'unisson synchrone via la satisfiabilité propositionnelle

Asma Khouldia, Sami Cherif, Stéphane Devismes, Léo Robert

Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

asma.khouldia@etud.u-picardie.fr, {sami.cherif;stephane.devismes;leo.robert}@u-picardie.fr

Résumé

L'unisson synchrone est un problème de synchronisation d'horloges de référence en système distribué, et notamment en autostabilisation. Cet article explore l'autostabilisation d'un algorithme d'unisson synchrone en utilisant une approche basée sur la satisfiabilité propositionnelle. Une formulation logique de l'algorithme est proposée, intégrant l'unicité des valeurs d'horloge des nœuds, leurs mises à jour en fonction de la valeur minimale des horloges des nœuds voisins, ainsi que la vérification de la convergence ou de la divergence. Afin d'optimiser la modélisation, des contraintes additionnelles sur les configurations initiales sont introduites pour limiter les cas redondants et non pertinents à analyser.

Mots-clés

Autostabilisation, unisson synchrone, satisfiabilité

Abstract

Synchronous unison is a benchmark clock synchronization problem in distributed computing, and especially in self-stabilization. This paper explores the self-stabilization of a synchronous unison algorithm using a propositional satisfiability-based approach. A logical formulation of the algorithm is proposed, incorporating the uniqueness of node clock values, their updates based on the minimum clock value at neighboring nodes, and the verification of the convergence or divergence. To optimize the modeling, additional constraints are introduced to reduce redundant and irrelevant cases to be analyzed.

Keywords

Self-stabilization, synchronous unison, satisfiability

1 Introduction

L'autostabilisation a été définie par Dijkstra [10] dans le contexte des systèmes distribués. Un système distribué est un ensemble d'entités de calcul, ici appelées *processus*, qui sont à la fois *autonomes* et *interconnectées* entre elles. Le but de ces processus est de coopérer afin de résoudre une tâche globale au système. Un algorithme est dit *autostabilisant* si, quel que soit l'état initial du système sur lequel il est déployé, il garantit le retour en temps fini, et sans intervention extérieure, à un état dit *légitime* à partir duquel

la spécification du système est réalisée. Bien que la notion de panne ne soit pas explicite dans la définition des systèmes autostabilisants, la motivation première de cette approche est pourtant la *tolérance aux pannes*. En effet, après un nombre fini de *pannes transitoires*¹, l'état d'un système distribué peut être quelconque et ainsi ne plus vérifier sa spécification. Un algorithme autostabilisant garantit alors que le système récupère de telles fautes en temps fini.

La conception d'algorithmes autostabilisants peut sembler complexe au premier abord en raison de l'accès limité des processus à l'état global du système. Pourtant, nombre de ces algorithmes se révèlent étonnamment élégants et parfois même plus simples que leurs pendants non stabilisants. Cependant, démontrer formellement l'autostabilisation d'un algorithme s'avère souvent ardu et sujet aux erreurs. Cela est principalement dû à la nature combinatoire du problème : la convergence vers un état légitime doit être démontrée à partir de *tout* état du système. Dans cet optique, nous nous intéressons ici à un problème simple : l'*unisson synchrone*. On suppose un système synchrone où les processus sont anonymes et arbitrairement connectés entre-eux. Chaque processus dispose d'une horloge locale dont les valeurs entières varient entre 0 à $m-1$ où m est appelée la *période*. À partir d'un état où l'horloge de chaque processus a une valeur quelconque de $\{0, \dots, m-1\}$, le but est de faire converger le système vers un état à partir duquel toutes les horloges sont identiques et s'incrémentent (modulo m) à chaque pas de calcul.

Arora et al. ont proposé une solution simple à ce problème [2]. L'autostabilisation de leur algorithme est démontrée pour tout $m \geq \max(2, 2D-1)$ où D est le diamètre du réseau [1, 2]. Cependant, l'optimalité de cette borne reste une question ouverte. Cette question est partiellement résolue dans [1] où des *états de divergence*, c'est-à-dire, des états à partir desquels des exécutions ne stabilisent jamais, sont exhibés pour toutes périodes *paires* inférieures strictement à la borne ; le cas impair restant non résolu à ce jour. Nous proposons ici d'étudier comment la satisfiabilité propositionnelle peut aider à traiter cette question, et plus généralement comment elle peut aider à vérifier l'autostabilisation d'algorithmes distribués.

1. C'est-à-dire, des perturbations temporaires de certains composants du systèmes (liens ou processus).

La *satisfiabilité* propositionnelle (SAT) est un problème fondamental en logique et en informatique théorique, consistant à déterminer si une formule booléenne peut être satisfaite par une assignation de valeurs. Ce problème, qui a été le premier démontré NP-complet [8], suscite un intérêt croissant car les solveurs SAT modernes parviennent à traiter efficacement de grandes instances issues d'applications concrètes, malgré la difficulté théorique établie du problème. En plus de son rôle clé en vérification de logiciels, d'intelligence artificielle et de cryptographie, SAT se situe à l'intersection de plusieurs disciplines telles que la logique, la complexité, la programmation par contraintes ou encore la théorie des graphes [3].

Dans cet article, nous proposons une approche formelle basée sur SAT pour analyser rigoureusement le comportement de l'unisson synchrone d'Arora et al. [2]. Les règles de mise à jour des horloges sont traduites en contraintes logiques sous forme normale conjonctive (FNC), permettant l'utilisation de solveurs SAT pour vérifier si la synchronisation est atteinte. Cette méthodologie offre un cadre systématique pour étudier la stabilisation de l'algorithme dans différentes topologies et configurations initiales et avec différentes périodes. Pour optimiser cette analyse, des techniques spécifiques sont appliquées pour réduire les redondances des configurations initiales, ce qui permet d'accélérer l'évaluation du modèle. Cette approche, bien qu'axée sur l'unisson, fournit des garanties formelles concernant les scénarios de synchronisation, tout en ouvrant des perspectives d'optimisation dans des systèmes distribués plus complexes.

Le reste de cet article est organisé comme suit. Dans la section 2, nous présentons l'algorithme d'unisson synchrone et nous introduisons la satisfiabilité propositionnelle. La section 3 est consacrée à la modélisation formelle de l'algorithme de l'unisson synchrone d'Arora et al. [2], où nous détaillons les variables et notations, ainsi que la formulation en forme normale conjonctive, avec une attention portée sur le maintien de l'unicité des horloges, la mise à jour des horloges et la modélisation la propriété d'autostabilisation, mais aussi la modélisation de contraintes additionnelles sur les configurations initiales dans le but d'accélérer la résolution du problème. Enfin, la section 4 présente une évaluation expérimentale des résultats obtenus. Nous concluons et discutons les perspectives de nos travaux en section 5.

2 Préliminaires

2.1 Unisson synchrone

Nous considérons un système distribué de n processus interconnectés. Le réseau d'interconnexion est modélisé par un graphe non-orienté connexe $G = (V, E)$ où V est un ensemble de n sommets représentant les processus et E un ensemble d'arêtes représentant les liens de communication bidirectionnels entre les processus. L'ensemble des voisins d'un processus p sera noté $N(p)$. L'unisson synchrone d'Arora et al. [2] est décrit dans le *modèle à états*, un modèle à mémoires localement partagées où chaque nœud peut directement lire son état et ceux de ses voisins, mais

peut uniquement modifier son propre état.

Étant donné un paramètre entier m commun à tous les processus et appelé *période*, chaque processus p dispose d'une seule variable $p.c \in \{0, \dots, m-1\}$, qui représente son *horloge*. La *configuration* du système est ainsi modélisée par le vecteur associant à chaque processus la valeur de son horloge. L'ensemble des configurations possibles du système sera noté $\Gamma_{n,m}$. L'algorithme local de chaque processus p est décrit par une unique règle de la forme *garde* $\rightarrow$ *action*. La garde de la règle de p est un prédicat sur l'état de p et ceux de ses voisins. La partie action est une affectation de l'horloge de p . Dans une configuration donnée, la règle d'un processus est dite activable si sa garde est vraie. Une configuration où aucune règle n'est activable est dite *terminale*.

Le système est supposé synchrone : tant que le système n'est pas dans une configuration terminale, des itérations sont exécutées comme suit. Tous les processus dont la règle est activable dans la configuration courante γ_i exécutent simultanément l'action de leur règle, générant ainsi une nouvelle configuration γ_{i+1} , et ainsi de suite. Une exécution de l'algorithme est donc une suite de configurations $\gamma_0, \dots, \gamma_i, \dots$ telle que pour tout $i > 0$, γ_i est obtenu à partir de γ_{i-1} en une itération synchrone de l'algorithme. Le but de l'algorithme est d'atteindre une configuration où toutes les horloges se sont synchronisées sur une valeur identique pour n'importe quelle configuration initiale possible.

Pour ce faire, à chaque itération, chaque processus p calcule la valeur d'horloge minimale $c_{\min}$ dans son voisinage clos (c'est-à-dire, $N(p) \cup \{p\}$), puis modifie, si nécessaire, son horloge $p.c$ à $(c_{\min} + 1) \bmod m$ (cf. Algorithme 1). De proche en proche, les valeurs les plus petites se propagent dans le réseau. Cette convergence peut uniquement être perturbée par l'apparition de 0 chez certains processus après que tout leur voisinage clos ait atteint la valeur maximale $m-1$. Arora et al. ont démontré que lorsque m est choisi suffisamment grand (c'est-à-dire, $m \geq \max(2, 2D-1)$ où D est le diamètre de G), ce souci de réinitialisation locale ne peut pas empêcher le système de converger puisque, quelle que soit la configuration initiale $\gamma_0 \in \Gamma_{n,m}$, il existe nécessairement une valeur $t \geq 0$ telle que toutes les horloges ont la même valeur dans γ_t suite à une exécution de l'algorithme.

Dans le cas où m est choisi trop petit, l'exécution peut

Algorithme 1 Unisson synchrone pour un processus p

Entrées :

$N(p)$: l'ensemble des voisins de p
 m : un entier positif, la période

Variable :

$p.c \in \{0, \dots, m-1\}$, l'horloge de p

Macro :

$New(p) = (\min(\{q.c : q \in N(p)\} \cup \{p.c\}) + 1) \bmod m$

Règle :

$p.c \neq New(p) \rightarrow p.c := New(p)$

ne jamais converger. Dans ce cas, il existe un cycle entre des configurations *illégitimes* (des configurations contenant au moins deux valeurs différentes). Or, le nombre total de configurations possibles est m^n . Donc, si après $m^n - 1$ itérations, le système est toujours dans une configuration illégitime, on peut affirmer que l'exécution ne convergera jamais. Ainsi, il est suffisant d'observer les $t_f = m^n$ premières configurations d'une exécution pour pouvoir décider si cette exécution converge ou diverge. Cependant, ce nombre est très grand et peut entraîner une modélisation exponentielle. On pourra alors choisir une valeur plus raisonnable pour t_f , quitte à ne pas pouvoir décider dans certains cas.

Exemple 1. Considérons un réseau de processus $G=(\{p_0, p_1, \dots, p_5\}, E)$ dont la topologie est en anneau. Posons $m = 5$ et supposons la configuration initiale γ_0 donnée dans la Figure 1. À partir de γ_0 , on obtient le préfixe d'exécution $\gamma_0, \gamma_1 \gamma_2, \gamma_3$ donné dans la Table 1. L'algorithme d'unisson atteint donc une configuration légitime après 3 itérations.

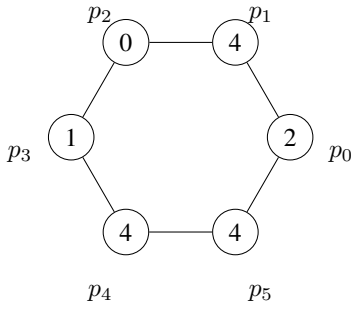


FIGURE 1 – Configuration initiale γ_0 dans un anneau de 6 nœuds avec $m = 5$.

	p_0	p_1	p_2	p_3	p_4	p_5
γ_0	2	4	0	1	4	4
γ_1	3	1	1	1	2	3
γ_2	2	2	2	2	2	3
γ_3	3	3	3	3	3	3

TABLE 1 – Préfixe de 4 configurations obtenu après 3 itérations depuis γ_0 .

Dans l'exemple précédent, une configuration légitime, où les horloges sont synchronisées est atteinte par l'algorithme. Cependant, lorsque m est mal choisi, il peut exister des configurations à partir desquelles les horloges ne se synchronisent jamais. On dira dans ce cas que le système *diverge*. Les propriétés de convergence et de divergence, formellement définies ci-dessous, mettent en évidence l'importance d'étudier le comportement des algorithmes distribués, et particulièrement l'unisson synchrone, et les paramètres influençant la synchronisation dans son contexte [9, 11].

Définition 1 (Convergence). Étant donné une période m , l'algorithme d'unisson synchrone exécuté sur un graphe G de n sommets converge lorsque pour toute configuration initiale $\gamma_0 \in \Gamma_{n,m}$, on atteint en un nombre fini d'itérations une configuration $\gamma_t \in \Gamma_{n,m}$ où toutes les valeurs d'horloge des sommets sont identiques.

Définition 2 (Divergence). Pour une période donnée m , l'algorithme d'unisson synchrone exécuté sur un graphe G de n sommets diverge lorsqu'il existe une configuration initiale $\gamma_0 \in \Gamma_{n,m}$ à partir de laquelle les horloges ne se synchronisent jamais : il existe une infinité de configurations atteintes contenant au moins deux valeurs d'horloge distinctes.

Exemple 2. Considérons un réseau de $n = 3$ processus dont la topologie est une chaîne. Posons $m = 2$ et considérons la configuration initiale γ_0 donnée dans la Figure 2. À partir de cette configuration initiale, on obtient le préfixe d'exécution décrit dans la Table 2. La divergence est donc atteinte dès la deuxième itération puisqu'on retrouve une configuration identique à γ_0 .

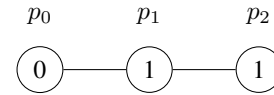


FIGURE 2 – Configuration initiale γ_0 dans une chaîne de 3 nœuds avec $m = 2$.

	p_0	p_1	p_2
γ_0	0	1	1
γ_1	1	1	0
γ_2	0	1	1

TABLE 2 – Deux itérations de l'unisson synchrone sur une chaîne de 3 nœuds avec $m = 2$.

Notre analyse va porter sur l'impact de la topologie du graphe et de la période m sur la synchronisation, en se concentrant sur des topologies simples comme les lignes, les anneaux et les étoiles. Le théorème suivant démontré dans [1] fournit notamment une limite générale intéressante pour le nombre maximum d'itérations dans un graphe connecté.

Théorème 1. L'algorithme d'unisson synchrone est autostabilisant dans tout graphe connecté de diamètre $\mathcal{D}$ pour $m \geq \max\{2, 2\mathcal{D} - 1\}$. Son temps de stabilisation est alors d'au plus $3\mathcal{D} - 2$ itérations.

2.2 Satisfiabilité propositionnelle

Soit X un ensemble des variables propositionnelles. Un littéral est une variable $x \in X$ ou sa négation $\bar{x}$. Une clause est une disjonction de littéraux. Une formule en Forme Normale Conjonctive (FNC) est une conjonction de clauses. Une affectation $I : X \rightarrow \{\text{vrai}, \text{faux}\}$ associe à

chaque variable une valeur booléenne et peut être représentée comme un ensemble de littéraux. Un littéral est satisfait par une affectation I si $l \in I$, sinon il est falsifié par I . Une clause est satisfaite par une affectation I si au moins un de ses littéraux est satisfait par I , sinon elle est falsifiée par I . Une formule FNC est satisfiable s'il existe une affectation qui satisfait toutes ses clauses, sinon elle est insatisfiable. Le problème de satisfiabilité (SAT) consiste à déterminer si une formule FNC donnée est satisfiable.

Bien que SAT soit NP-complet [8], les solveurs modernes basés sur l'algorithme CDCL (Conflict Driven Clause Learning) [12] sont efficaces et parviennent à résoudre des instances impliquant un grand nombre de variables et de clauses. En effet, au delà de l'apprentissage de clauses et du retour arrière non chronologiques, ces solveurs intègrent des mécanismes puissants comme, par exemple, les structures paresseuses, des heuristiques de branchement dédiées ou encore les redémarrages [6]. Par ailleurs le problème SAT est largement utilisé pour modéliser et résoudre de nombreux problèmes combinatoires complexes issus de divers domaines, notamment en vérification formelle [4, 7] mais aussi en cryptographie, en bio-informatique et en planification [6].

Dans la suite, on s'intéresse à modélisation du comportement de l'algorithme d'unisson synchrone, appliqué à un graphe arbitrairement connecté. L'objectif est de caractériser les conditions de convergence ou de divergence de l'algorithme. Cela implique la modélisation des états des horloges des nœuds du réseau à l'aide de formules logiques, qui doivent être exprimées de manière à pouvoir être résolues efficacement par un solveur SAT.

3 Modélisation formelle de l'unisson synchrone

Dans cette section, nous nous intéressons à la modélisation formelle du comportement de l'unisson synchrone à travers la satisfiabilité propositionnelle. Nous commençons par introduire les variables du modèle, puis nous définissons les contraintes qui encadrent l'exécution de l'algorithme. Ces contraintes doivent non seulement représenter fidèlement l'évolution du système, mais aussi permettre de détecter sa convergence ou sa divergence. On s'intéressera également à l'introduction de contraintes permettant d'optimiser le modèle en éliminant des configurations initiales redondantes.

3.1 Variables et Notations

On rappelle que le réseau est modélisé par un graphe non-orienté connexe $G = (V, E)$ où $V = \{0, \dots, n-1\}$ est un ensemble de n sommets représentant les processus. On notera $M = \{0, \dots, m-1\}$ l'ensemble des valeurs d'horloges possibles, où m est la période. Nous nous intéressons aux t_f premières configurations d'une exécution quelconque de l'algorithme sur G . Elles seront indicées de 0 à $t_f - 1$. L'ensemble des indices sera noté $T = \{0, \dots, t_f - 1\}$. Pour modéliser l'exécution de l'algorithme, il est aussi nécessaire de raisonner sur l'état des horloges des processus dans les différentes configurations. Pour cela, on introduit les variables

d'horloge $h_{p,t,v}$, pour tout $(p, t, v) \in V \times T \times M$. Chaque variable $h_{p,t,v}$ indiquera si l'horloge du processus p est égale à la valeur v dans la configuration t . Cela s'avère suffisant pour modéliser l'exécution et pour détecter la convergence. Cependant, la modélisation de la divergence nécessite d'autres variables, notamment pour pouvoir représenter la présence d'un cycle entre configurations illégitimes pendant l'exécution et pour simplifier la transformation des contraintes en forme normale conjonctive. L'ensemble des variables qui seront utilisées est donné ci-dessous :

- **Variables d'horloge** : $h_{p,t,v}$ où $p \in V, t \in T, v \in M$. Ces variables sont affectées à *vrai* si le processus p a la valeur d'horloge v dans la configuration t et à *faux* sinon.
- **Variables de cycle** : c_t où $t \in T$. Ces variables sont affectées à *vrai* si la configuration $t \in T$ et la configuration initiale sont identiques.
- **Variables de similarité d'horloge** : $s_{p,t,v}$ où $p \in V, t \in T$ et $v \in M$. Ces variables seront affectées à *vrai* si le processus p a la même valeur d'horloge dans la configuration $t \in T$ et la configuration initiale.

3.2 Modélisation de l'exécution de l'unisson

Dans un premier temps, nous nous intéressons à la formalisation des contraintes modélisant une exécution valide de l'unisson synchrone. Plus précisément, nous définissons les contraintes d'unicité des horloges ainsi que les règles de mise à jour assurant la propagation correcte des valeurs d'horloge au fil des itérations.

3.2.1 Unicité des horloges

Pour garantir une exécution valide de l'algorithme, il est nécessaire de maintenir l'unicité de l'horloge de chaque processus à chaque itération. Précisément, il faut veiller à ce que chaque processus possède une et une seule valeur d'horloge à chaque configuration de l'algorithme. Cette contrainte peut être modélisée comme suit :

$$\bigwedge_{\substack{(v,v') \in M^2 \\ v < v'}} (\overline{h_{i,t,v}} \vee \overline{h_{i,t,v'}}) \quad \forall p \in V, \forall t \in T \quad (1a)$$

$$\bigvee_{v \in M} h_{p,t,v} \quad \forall p \in V, \forall t \in T \quad (1b)$$

La contrainte (1a) garantit qu'un processus ne peut pas posséder simultanément deux valeurs d'horloge différentes dans une configuration donnée tandis que la contrainte (1b) assure que chaque processus possède au moins une valeur d'horloge valide à chaque configuration. Le nombre de clauses induit par ces contraintes est en $O(n * t_f * m^2)$.

3.2.2 Mise à jour des horloges

Dans le cadre de l'algorithme de l'unisson synchrone, les horloges des processus doivent être mises à jour, à chaque itération, en fonction des valeurs d'horloge de leurs voisins immédiats. Plus précisément, la nouvelle valeur d'horloge du processus p est déterminée en prenant le minimum des

valeurs d'horloge de ses voisins, ainsi que la valeur d'horloge qu'il détenait à l'étape précédente. Cette valeur sera ensuite incrémentée de 1 modulo m . Pour simplifier les notations, on dénote $\tilde{N}(p) = N(p) \cup \{p\}$ le *voisinage clos* du processus $p \in V$. La contrainte de mise à jour des horloges est alors modélisée comme suit :

$$\bigwedge_{\substack{(v_{p'})_{p' \in \tilde{N}(p)} \in M^{d_p} \\ v' = (\min_{p' \in \tilde{N}(p)} v_{p'} + 1) \bmod m}} \left(\bigvee_{p' \in \tilde{N}(p)} \overline{h_{p',t,v_{p'}}} \right) \vee h_{p,t+1,v'} \quad \forall t \in T \setminus \{t_f - 1\}, \forall p \in V \quad (2)$$

En effet, pour chaque assignation possible $(v_{p'})_{p' \in \tilde{N}(p)}$ des valeurs d'horloges du voisinage clos d'un processus p , on veut s'assurer que la valeur minimale incrémentée de 1 modulo m , dénotée v' , soit propagée au processus p à la configuration suivante, c'est à dire $\bigwedge_{p' \in \tilde{N}(p)} h_{p',t,v_{p'}} \rightarrow h_{p,t+1,v'}$. Cette dernière implication met parfaitement en évidence le mécanisme de mise à jour des horloges dans le cadre de l'unisson synchrone et est donc réécrite sous forme normale conjonctive dans la contrainte (2), en rajoutant les itérations nécessaires sur les assignations possibles et le calcul de la nouvelle valeur de l'horloge au niveau de la conjonction. Pour fournir une analyse fine de la complexité de l'encodage, on note $d_{\max} = \max_{p \in V} |N(p)|$ le degré maximal du graphe. Ainsi, il est clair que le nombre de clauses serait borné par $O(n * t_f * m^{d_{\max}+1})$. Il semblerait donc que la complexité de notre modèle soit fortement corrélée au degré maximal dans le graphe et qu'il soit donc plus adapté aux graphes pour lesquels cette valeur reste relativement petite. En particulier, pour les graphes en anneau ou en chaîne la complexité sera en $O(n * t_f * m^3)$ puisque, pour ces topologies, on a $d_{\max} = 2$.

3.3 Modélisation de la convergence

La convergence se produit lorsque les valeurs d'horloge des processus atteignent en un nombre fini d'itérations une configuration *légitime*, où toutes les valeurs d'horloge sont identiques pour l'ensemble des processus. Un algorithme est dit convergent si, à partir de n'importe quelle configuration initiale, une telle configuration légitime est toujours atteinte en un nombre fini d'itérations. Pour démontrer cette propriété, nous allons raisonner par l'absurde en supposant que la convergence n'est jamais atteinte à partir d'une certaine configuration initiale. Autrement dit, il existe (au moins) une configuration initiale, à partir de laquelle aucune configuration légitime ne peut être obtenue. Étant donné qu'une configuration légitime engendre nécessairement une autre configuration légitime après une itération de l'algorithme, il suffit alors de montrer que la dernière configuration $t_f - 1$ n'est pas légitime pour une certaine configuration initiale. Cette propriété est ainsi traduite par la contrainte suivante :

$$\bigwedge_{v \in M} \bigvee_{p \in V} \overline{h_{p,t_f-1,v}} \quad (3)$$

Ainsi, il est clair que la formule CNF ϕ obtenue grâce aux contraintes (1), (2) et (3) est insatisfiable si et seulement si l'algorithme d'unisson synchrone converge. En effet, si ϕ est satisfiable, le modèle fourni permet d'exhiber une configuration initiale $\lambda_0 \in \Gamma_{n,m}$ déduite des variables d'horloge $h_{p,0,v}$ affectées à *vrai* pour $p \in V$ et $v \in M$ à partir de laquelle l'algorithme ne converge pas. Plus spécifiquement, après une exécution valide de l'algorithme d'unisson à partir de λ_0 (assurée par les contraintes (1) et (2)), on arrive toujours à exhiber, au niveau de la dernière configuration t_f et pour toute valeur d'horloge possible $v \in M$, un processus $p \in V$ dont la valeur d'horloge ne prend pas la valeur v dans t_f (assuré par la contrainte (3)). En terme de complexité, la contrainte (3) nécessitant seulement $O(m)$ clauses, notre modèle pour la convergence nécessite seulement $O(n \times t_f \times m)$ variables avec un nombre de clauses borné en $O(n * t_f * m^{d_{\max}+1})$ où $d_{\max}$ est le degré maximal du graphe.

3.4 Modélisation de la divergence

La divergence implique l'existence de deux configurations illégitimes identiques dans l'exécution. En effet, l'ensemble des configurations légitimes est trivialement clos et l'algorithme étant synchrone et déterministe, un tel cycle se répète à l'infini. Soit deux configurations $(t_1, t_2) \in T^2$ tels que $t_1 < t_2$, où les valeurs d'horloge des processus en t_2 sont identiques à celles en t_1 . On appellera une telle occurrence un *cycle* au niveau des configurations (t_1, t_2) . Pour établir la divergence, il faut donc modéliser l'existence d'un cycle au niveau de deux configurations illégitimes possibles atteintes par l'algorithme. Pour simplifier la modélisation, on considère l'existence de cycle par rapport à la configuration initiale, c'est à dire $t_1 = 0$. En effet, même si $t_1 > 0$, il est toujours possible de ramener l'analyse à la configuration initiale en prenant la même configuration comme état initial du système (*n.b.*, un algorithme autostabilisant doit converger depuis n'importe quelle configuration initiale). De plus, pour garantir que les configurations du cycle soient illégitimes, il suffit simplement de vérifier que la configuration initiale est illégitime. On rappelle que les variables c_t où $t \in T$, définies auparavant, représentent l'existence de cycle à la configuration t . Ainsi l'existence d'un cycle entre configurations illégitimes peut se traduire par les contraintes suivantes :

$$\bigvee_{\substack{t \in T \\ 0 < t}} c_t \quad (4a)$$

$$\bigwedge_{v \in M} \bigvee_{p \in V} \overline{h_{p,0,v}} \quad (4b)$$

Clairement, la contrainte (4a) modélise l'existence d'un cycle $(0, t)$ où $t \in T$; tandis que la contrainte (4b) garantit que la configuration 0, et donc toutes les configurations du cycles, sont illégitimes. Pour instaurer le sens sémantique de variables de cycle, il faut garantir que, si ses variables sont affectées à *vrai*, alors il y a un cycle $(0, t)$, c'est à dire que les valeurs d'horloge dans la configuration t

sont identiques à celles dans la configuration initiale. Dans ce contexte, les variables de similarité d'horloge $s_{p,t,v}$ où $p \in V$, $t \in T$ et $v \in M$ peuvent jouer le rôle intermédiaire de garantir une même valeur d'horloge dans la configuration $t \in T$ et la configuration initiale. Ainsi, on rajoute les contraintes suivantes :

$$\overline{c_t} \vee \bigvee_{v \in M} s_{p,t,v} \quad \forall t \in T, \forall p \in V \quad (5a)$$

$$(\overline{s_{p,t,v}} \vee h_{p,0,v}) \wedge (\overline{s_{p,t,v}} \vee h_{p,t,v}) \quad \forall t \in T, \forall p \in V, \forall v \in M \quad (5b)$$

La contrainte (5a) assure que la configuration $t \in T$ correspondante a les mêmes valeurs d'horloge que la configuration 0 ou, plus formellement, que $c_t \rightarrow \bigwedge_{p \in V} \bigvee_{v \in M} s_{p,t,v}$. La forme normale présentée dans la contrainte (5a) peut donc être simplement obtenue par la réécriture clausale de l'implication ainsi que la distributivité des disjonctions sur les conjonctions logiques. La contrainte (5b) est nécessaire pour instaurer le sens sémantique des variables $s_{p,t,v}$ qui sont affectées à *vrai* quand le processus $p \in V$ a la même valeur d'horloge $v \in V$ aux configurations 0 et t . Plus formellement, on a $s_{p,t,v} \rightarrow h_{p,0,v} \wedge h_{p,t,v}$ et sa forme normale présentée dans la contrainte (5b) peut également être déduite par réécriture de l'implication et distributivité de la disjonction.

Ainsi, il est clair que la formule CNF ψ obtenue par les contraintes (1), (2), (4) et (5) est satisfiable si et seulement si l'algorithme d'unisson diverge sur le réseau donné en entrée avec la période m . En effet, un modèle de ψ permet d'exhiber une configuration initiale $\lambda_0 \in \Gamma_{n,m}$ à partir de la quelle une exécution valide de l'unisson synchrone génère un cycle $(0, t)$ de configurations illégitimes, donné par une variable de cycle affectée à *vrai* au niveau de la contrainte clausale (4a). De plus, en terme de complexité de l'encodage pour la divergence, il est clair que les contraintes (4) et (5) introduisent un nombre de clauses borné en $O(n \times t_f \times m)$ et, par conséquent, que le modèle a une complexité similaire au cas de la convergence, principalement régie par la contrainte (2), c'est à dire en $O(n * t_f * m^{d_{max}+1})$ où d_{max} est le degré maximal du graphe. Cependant, le modèle nécessite un nombre plus grand de variables, mais qui reste borné en $O(n \times t_f \times m)$.

3.5 Élimination de configurations initiales

Dans cette section, nous cherchons à réduire le nombre de configurations initiales à traiter en supprimant les configurations redondantes. En d'autres termes, il s'agit d'éliminer les configurations pour lesquelles le comportement de l'algorithme est identique à celui d'autres configurations déjà prises en compte. Cette approche permet d'optimiser l'exécution de l'algorithme en évitant de traiter des situations qui, bien que différentes dans leur représentation, ne conduisent pas à des résultats différents.

3.5.1 Ordre lexicographique

Dans le cadre de l'algorithme d'unisson, l'évolution des valeurs d'horloge des processus dépend exclusivement de

leur voisinage local et, plus particulièrement, de leur voisinage clos. En particulier, deux processus possédant le même ensemble de voisins dans la topologie étudiée évoluent de manière identique, à permutation près de leurs valeurs d'horloge initiales. Cette observation permet d'exploiter une symétrie structurelle afin de réduire l'espace des configurations à considérer en instaurant un ordre lexicographique particulier sur les valeurs initiales d'horloge pour des processus ayant un voisinage identique dans la topologie. Pour tirer parti de cette symétrie et éviter d'examiner plusieurs configurations équivalentes, nous introduisons des contraintes imposant un ordre lexicographique sur les configurations. Plus précisément, seules les configurations minimales selon cet ordre sont considérées, ce qui permet d'éliminer les configurations redondantes et de réduire l'espace de recherche. Dans notre cas, on va introduire des contraintes interdisant les configurations lexicographiquement plus grandes sur les voisinages locaux identiques que celles sélectionnées. Cette contrainte, qu'on dénotera OL, peut être formulée comme suit :

$$\bigwedge_{\substack{(p,p') \in V^2 \\ p < p' \\ N(p)=N(p')}} \bigwedge_{\substack{(v,v') \in M^2 \\ v' < v}} (\overline{h_{p,0,v}} \vee \overline{h_{p',0,v'}}) \quad (6)$$

Ainsi, si deux processus $(p, p') \in V^2$ tels que $p < p'$ ont un voisinage identique, la contrainte (6) garantit que p' ne puisse pas prendre une valeur d'horloge initiale plus petite que celle de p . Cette contrainte introduit un nombre de clauses borné en $O(n^2 \times m^2)$ et ne peut être appliquée que dans des topologies où certains processus possèdent un voisinage identique, comme dans la topologie en étoile, où un processus central (hub) est connecté à plusieurs processus périphériques (voir Figure 4), ou encore dans les graphes complets.

3.5.2 Élimination des rotations

Comme éayé précédemment, lorsqu'une transformation d'une configuration initiale ne modifie pas le comportement global de l'algorithme, il est possible d'éliminer les configurations redondantes afin d'optimiser la recherche et l'analyse des cas pertinents. Un cas particulièrement intéressant concerne les graphes, tels que les anneaux, où une même configuration peut être exprimée sous plusieurs formes équivalentes via une simple rotation des indices des processus. Dans ces structures, toute configuration initiale peut être transformée en une autre configuration par un décalage uniforme des valeurs d'horloge. Dès lors que la rotation ne modifie ni les relations entre voisins ni la dynamique de l'algorithme, on peut restreindre l'étude à un sous-ensemble représentatif des configurations possibles.

Afin d'imposer cette restriction, on peut introduire une contrainte garantissant que, parmi toutes les configurations équivalentes, seules celles où un processus spécifique détient la plus faible valeur d'horloge sont considérées. En particulier, dans un graphe en anneau de n processus, on considère une configuration où le premier processus possède la valeur d'horloge la plus faible parmi les valeurs

d'horloge des autres processus dans la même configuration. Puisque la rotation, qu'elle soit dans le sens direct ou indirect, conserve le même comportement de l'algorithme d'unisson synchrone, on éliminera toute autre configuration initiale qui est image de cette configuration afin d'éviter l'analyse de cas redondants, comme démontré dans la proposition 1, avec $\lambda[p]$ qui désigne la valeur d'horloge de $p \in V$ dans la configuration $\lambda \in \Gamma_{n,m}$. Cette contrainte, qu'on dénotera ER, peut donc être exprimée comme suit sur toute topologie en anneau :

$$\bigwedge_{\substack{(v,v') \in M^2 \\ v < v'}} \bigwedge_{p \in V \setminus \{0\}} (\overline{x_{0,0,v'}} \vee \overline{x_{p,0,v}}) \quad (7)$$

Proposition 1. Soit $G = (V, E)$ un graphe en anneau composé de n processus et m une période. Soit $ER = \{\lambda \in \Gamma_{n,m} \mid \exists p \in V \setminus \{0\}, \lambda[p] < \lambda[0]\}$. Alors, pour toute configuration initiale $\lambda_0 \in ER$, il existe une configuration $\lambda'_0 \in \Gamma_{n,m} \setminus ER$ à partir de laquelle une exécution de l'unisson synchrone produit le même comportement de convergence ou de divergence.

Démonstration. Sans perte de généralité, on suppose que $V = \{0, \dots, n-1\}$. Soit $\lambda_0 \in ER$ et $p_{\min} = \arg\min_{p \in V} \lambda_0[p]$. Nous définissons la configuration λ'_0 telle que :

$$\lambda'_0[p] = \begin{cases} \lambda[p - p_{\min}] & \text{pour } p_{\min} \leq p \leq n-1, \\ \lambda[n - p_{\min} + p] & \text{pour } 0 \leq p < p_{\min}. \end{cases}$$

Clairement, λ'_0 est obtenue par une rotation de λ_0 qui ramène la valeur la plus petite au processus 0. Donc, λ'_0 a le même comportement que λ , à rotation près, suite à une exécution de l'algorithme d'unisson synchrone puisque, sous rotation, les relations entre voisins sont préservées et chaque processus applique exactement les mêmes règles d'évolution que dans la configuration originale. Par ailleurs, puisque $\lambda'_0[0] = \lambda_0[p_{\min}] = \min_{p \in V} \lambda[p]$, il n'existe pas de processus $p \in V \setminus \{0\}$ tel que $\lambda[p] < \lambda[0]$ et donc λ'_0 appartient bien au noyau $\Gamma_{n,m} \setminus ER$. $\square$

La contrainte (7) introduit un nombre de clauses borné en $O(n \times m^2)$ et peut donc être appliquée sur les graphes dont la topologie permet de faire des rotations particulières en préservant les relations entre voisins comme la topologie en anneau ou encore la topologie en étoile en appliquant la contrainte uniquement sur les processus périphériques.

4 Évaluation expérimentale

4.1 Protocole expérimental

Nous avons encodé nos modèles en Python en utilisant la bibliothèque PySAT². Nous avons généré une instance pour chaque type de topologie (étoiles, chaînes et anneaux), chaque nombre de nœuds et chaque période. Nous avons résolu ces instances en utilisant le solveur Cadical [5]. Les

tests ont été réalisés sur la plateforme MatriCS³, sur une machine équipée d'un processeur Intel Core i7 cadencé à 3,80 GHz, sous Ubuntu 22.04. Un temps limite de 3600 s a été considéré pour chaque instance.

Nous avons généré un total de 720 instances réparties comme suit. Pour chaque type de topologie parmi les anneaux, les chaînes et les étoiles, on prend des tailles de graphes n allant de 3 jusqu'à 10 nœuds, et on fait varier les périodes m de 2 à 10. Ainsi, on obtient 72 formules de convergence (resp. divergence) avec le modèle initial (INI) incluant les contraintes (1), (2) et (3) (resp. (1), (2), (4) et (5)) pour chaque topologie. Par ailleurs, pour les topologies permettant des contraintes d'élimination de configurations initiales, on génère également les instances correspondantes. Ainsi, on a généré 144 instances de convergence et de divergence pour la topologie en anneau (resp. étoile) avec la contrainte d'élimination des rotation (resp. d'ordre lexicographique). Enfin, on fixe $t_f = 100$ pour toutes les instances.

4.2 Analyse de la convergence et la divergence de l'unisson synchrone

Les résultats présentés dans la Figure 3 illustrent le comportement de l'algorithme d'unisson synchrone en termes de convergence et de divergence sur les différents types de topologies (anneaux, chaînes et étoiles) en fonction de la taille n et de la période m . Pour les anneaux et les chaînes, toutes les instances ont pu être résolues, révélant une convergence alignée avec le Théorème 1 lorsque $m \geq \max\{2, 2D - 1\}$. On rappelle que ces topologies présentent des diamètres qui évoluent en fonction de la taille des graphes avec $D = \lfloor n/2 \rfloor$ pour les anneaux et $D = n-1$ pour les chaînes.

En revanche, pour les graphes en étoile, certaines instances n'ont pas été résolues dans la limite d'une heure. Ceci est traduit par la complexité accrue du modèle dans le cadre de la topologie en étoile, à cause du nœud central dont le voisinage inclut tous les nœuds périphériques. Les résultats expérimentaux en terme de performance confirmeront également cette observation dans la section 4.3. Cependant, il est intéressant de noter que nos résultats ont montré que la borne proposée dans Théorème 1 est précise dans le cas des étoiles. En effet, ce théorème indique que la convergence est garantie dans les étoiles (de diamètre 2) à partir d'une période m supérieure ou égale à 3 ($\max(2, 2 \times 2 - 1) = 3$). Or, nos résultats exhibent un cas de divergence avec $m = 2$ dans les graphes en étoiles, indépendamment de leurs tailles. Ainsi, l'algorithme d'unisson synchrone converge dans une étoile si et seulement si $m \geq 3$. Nous démontrons que notre contre-exemple pour $m = 2$ est général dans les étoiles dans la proposition suivante.

Proposition 2. L'algorithme d'unisson synchrone diverge pour tout graphe en étoile, quel que soit son nombre de nœuds $n \geq 3$, avec la période $m = 2$.

2. <https://pysathq.github.io/>

3. <https://www.matrics.u-picardie.fr/catalogue-de-services/compte-matrics/>

10 9 8 7 6 5 4 3 2	m n	(a) Anneaux							
		3	4	5	6	7	8	9	10
10		✓	✓	✓	✓	✓	✓	✓	✓
9		✓	✓	✓	✓	✓	✓	✓	✓
8		✓	✓	✓	✓	✓	✓	✓	✓
7		✓	✓	✓	✓	✓	✓	✓	✓
6		✓	✓	✓	✓	✓	✓	✓	✓
5		✓	✓	✓	✓	✓	✓	✓	✓
4		✓	✓	✓	✓	✓	✓	✓	✓
3		✓	✓	✓	✓	✓	✓	✓	✓
2		✓	✓	✓	✓	✓	✓	✓	✓

10 9 8 7 6 5 4 3 2	m n	(b) Chaînes							
		3	4	5	6	7	8	9	10
10		✓	✓	✓	✓	✓	✓	✓	✓
9		✓	✓	✓	✓	✓	✓	✓	✓
8		✓	✓	✓	✓	✓	✓	✓	✓
7		✓	✓	✓	✓	✓	✓	✓	✓
6		✓	✓	✓	✓	✓	✓	✓	✓
5		✓	✓	✓	✓	✓	✓	✓	✓
4		✓	✓	✓	✓	✓	✓	✓	✓
3		✓	✓	✓	✓	✓	✓	✓	✓
2		✓	✓	✓	✓	✓	✓	✓	✓

10 9 8 7 6 5 4 3 2	m n	(c) Étoiles							
		3	4	5	6	7	8	9	10
10		✓	✓	✓	✓	-	-	-	-
9		✓	✓	✓	✓	-	-	-	-
8		✓	✓	✓	✓	-	-	-	-
7		✓	✓	✓	✓	✓	-	-	-
6		✓	✓	✓	✓	✓	✓	-	-
5		✓	✓	✓	✓	✓	✓	✓	-
4		✓	✓	✓	✓	✓	✓	✓	✓
3		✓	✓	✓	✓	✓	✓	✓	✓
2		✓	✓	✓	✓	✓	✓	✓	✓

FIGURE 3 – Résultats de convergence et de divergence de l'unisson synchrone sur les anneaux, chaînes et étoiles, pour différentes tailles de graphe n et périodes m . Les signes (✓), (⊖) et (−) indiquent respectivement que l'algorithme converge, diverge ou que l'instance n'a pas été résolue dans le temps alloué.

Démonstration. Nous pouvons toujours exhiber une configuration initiale $\gamma_0 \in \Gamma_{n,m}$ qui entraîne la divergence de l'unisson synchrone. Une telle configuration est illustrée dans la Figure 1, où la valeur initiale de l'horloge du nœud central est fixée à 1, tandis que les valeurs des horloges des nœuds périphériques prennent au moins deux valeurs initiales distinctes. Sans perte de généralité, nous supposons que tous les nœuds ayant une valeur 0 sont situés à gauche du nœud central, tandis que les nœuds ayant une valeur d'horloge égale à 1 sont situés à sa droite.

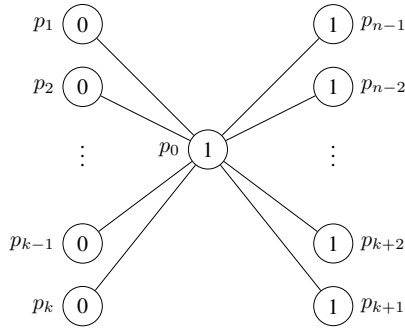


FIGURE 4 – Configuration initiale assurant la divergence de l'unisson synchrone sur un graphe en étoile avec n nœuds et en fixant la période à $m = 2$.

Lors de l'exécution de l'algorithme d'unisson synchrone sur un tel graphe avec la configuration initiale correspondante, le nœud central mettra à jour son horloge à 1, les nœuds périphériques situés à gauche mettront également à jour leurs horloges à 1, tandis que les nœuds périphériques situés à droite mettront à jour leurs horloges à 0. À l'itération suivante, un cycle de configurations illégitimes est alors atteint, car nous revenons à la configuration non convergente γ_0 . Étant donné que ce comportement se maintient pour un graphe en étoile de toute taille, nous concluons que l'unisson synchrone diverge sur de tels graphes, indépendamment de leurs tailles, lorsque $m = 2$. $\square$

4.3 Analyse des performances

Dans cette section, nous analysons les performances des modèles sur différentes topologies en termes de temps

moyen de résolution par taille de graphe. Les résultats obtenus pour les graphes en anneau, en chaîne et en étoile sont présentés dans le Tableau 3, en fonction de la taille du graphe, de la propriété analysée et du modèle considéré. Les résultats indiquent que le modèle initial (INI) parvient à résoudre l'ensemble des instances générées pour les graphes en anneau et en chaîne. En revanche, ce même modèle rencontre des difficultés croissantes avec la taille du graphe pour la topologie en étoile. Il parvient ainsi à résoudre 51 (resp. 50) instances sur 72 pour la vérification de convergence (resp. divergence) sur les étoiles. Ce comportement n'est pas surprenant, car cette topologie présente un degré maximal $d_{\max} = n - 1$, ce qui entraîne une explosion combinatoire du nombre de clauses générées par la contrainte (2). En effet, celui-ci croît selon une complexité de l'ordre de $O(n \times t_f \times m^n)$, ce qui explique la difficulté du modèle à gérer les grandes instances.

Les Figures 5 et 6 illustrent l'évolution du temps de résolution des instances générées pour la vérification de la divergence, en fonction de la taille du graphe n et de la période m , respectivement pour les topologies en anneau et en étoile. Ces courbes mettent en évidence une augmentation progressive du temps de résolution à mesure que ces deux paramètres augmentent. On observe plus particulièrement que, dans le cas des graphes en étoile, cette croissance est plus marquée que pour les graphes en anneau. Cette tendance, qui reste similaire pour la convergence ainsi que pour les graphes en chaîne, corrobore nos observations antérieures et confirme la complexité accrue du modèle pour certaines topologies.

Dans la suite, nous étudions les formulations augmentées par les contraintes d'élimination des configurations initiales. Comme l'illustre le Tableau 3, la comparaison avec le modèle initiale met en évidence la pertinence d'introduire ces contraintes, en particulier pour l'analyse de la divergence. Les résultats montrent une réduction significative du temps de résolution pour les topologies en anneau et en étoile grâce aux contraintes d'élimination de rotation (ER) et d'ordre lexicographique (OL), respectivement. Plus précisément, pour les graphes en anneau, le modèle ER atteint les meilleures performances globales, avec un temps total de résolution de 18.1s pour la convergence et 57.61s pour la divergence, sur l'ensemble des tailles de graphes testées.

n	Anneaux				Chaînes		Étoiles			
	CONV		DIV		CONV	DIV	CONV		DIV	
	INI	ER	INI	ER	INI	INI	INI	OL	INI	OL
3	0.01	0.01	0.60	0.31	0.01	0.36	0.006	0.006	0.26	0.16
4	0.18	0.11	1.96	0.60	0.10	1.4	0.15	0.33	3.10	1.29
5	0.53	0.52	4.88	1.58	0.34	3.73	5.04	5.78	46.10	11.90
6	1.13	1.13	11.07	3.17	0.53	7.03	123.25	176.65	483.09 [8]	211.90
7	1.71	2.07	19.72	5.45	2.11	8.01	102.23 [6]	89.43 [6]	591.61 [6]	92.33 [6]
8	3.19	2.72	35.30	9.72	3.20	14.78	48.76 [4]	52.00 [4]	351.62 [4]	39.56 [4]
9	4.54	3.93	64.80	16.88	4.57	21.39	44.89 [3]	56.34 [3]	188.65 [3]	15.61 [3]
10	8.11	7.61	59.59	19.90	7.94	28.07	7.58 [2]	0.81 [2]	28.73 [2]	1.63 [2]

TABLE 3 – Temps de résolution en secondes pour les topologies en anneau, chaîne et étoile. Pour chaque topologie, les temps moyens de résolution pour la détection de la convergence (CONV) et la divergence (DIV) sur toutes les périodes pour une taille de graphe donnée (n) sont reportés. Les résultats des modèles optimisés sont également reportés pour les topologies permettant des contraintes supplémentaires (ER pour les anneaux et OL pour les étoiles) et les meilleurs résultats entre les modèles initial et optimisé sont marqué en gras. Pour une taille n donnée avec des instances ayant atteint le temps limite, le nombre d’instances résolues est indiqué entre ‘[]’.

Comparé au modèle initial, ER permet ainsi de réduire le temps de résolution total de 6.701% pour la convergence et de 70.892% pour la divergence. Ces résultats sont aussi mis en évidence dans la Figure 5, démontrant ainsi l’impact positif de cette optimisation sur l’analyse du comportement de l’unisson synchrone. Le graphe en étoile révèle une dynamique similaire puisque le modèle OL démontre une amélioration significative du temps de divergence avec une valeur totale de 374.38s, soit une réduction de 77.858% par rapport au modèle initial (1 693.16s). Ces observations sont aussi confirmées par la Figure 5 et mettent en avant la pertinence de cette optimisation dans des structures où les interactions sont fortement centralisées autour d’un nœud pivot. Toutefois, pour la convergence, le modèle OL reste légèrement en retrait par rapport au modèle ER, ce qui suggère que son efficacité se manifeste principalement dans la vérification de la divergence.

Pour résumer, les résultats montrent que, même sans optimisation spécifique, les modèles sont capables d’analyser le comportement de l’unisson synchrone, notamment pour les graphes en chaîne qui présentent déjà des temps de résolution compétitifs. Les modèles optimisés, en particulier ER, se révèlent être les plus efficaces pour réduire l’efficacité du modèle dans les graphes en anneau, tandis que OL se distingue par une performance meilleure lors de la vérification de la divergence dans les graphes en étoile, d’autant plus que ce modèle a permis de résoudre une instance de plus par rapport au modèle initial.

5 Conclusion

Dans cet article, nous avons présenté une approche formelle basée sur la satisfiabilité propositionnelle (SAT) pour analyser le comportement de un algorithme autostabilisant d’unisson synchrone. Précisément, on s’est intéressé à la détection des cas où cet algorithme converge ou diverge. En représentant les états du problème et son exécution en contraintes logiques, nous avons montré comment les solveurs SAT peuvent être utilisés pour prouver l’autostabili-

sation d’un système distribué ou, dans le cas contraire, pour produire un contre-exemple. Ce cadre a permis d’étudier rigoureusement l’autostabilisation d’un algorithme d’unisson synchrone dans des topologies variées, en prenant en compte différentes périodes. En outre, nous avons optimisé l’analyse en proposant des contraintes spécifiques pour éliminer les configurations initiales redondantes, ce qui a permis d’accélérer l’évaluation du modèle et de garantir des résultats plus efficaces. Nos modèles ont également permis d’exhiber des comportements particuliers, notamment la divergence dans le cas des étoiles quand $m = 2$.

Plusieurs pistes de recherche restent ouvertes. Tout d’abord, l’extension de notre analyse à d’autres types de graphes, tels que les grilles, les tores ou les arbres binaires, permettrait d’examiner plus en détail l’impact d’autres topologies distribuées classiques sur l’autostabilisation de l’algorithme. Ces structures, particulièrement pertinentes pour des applications concrètes, offrent une grande diversité de configurations qui méritent une investigation approfondie afin d’affiner notre compréhension des dynamiques de convergence et de divergence. On pourra aussi affiner le choix du nombre maximal de configurations autorisées par rapport à la topologie étudiée.

Il serait également pertinent d’étudier d’autres formulations plus compactes de la contrainte de mise à jour des horloges afin de réduire la complexité du modèle. De plus, pour la contrainte de convergence, il serait possible d’intégrer une analyse plus fine en instaurant des contraintes redondantes sur l’ensemble des configurations au lieu de se focaliser uniquement sur la vérification de la synchronisation au niveau de la dernière. Une telle approche permettrait alors de détecter plus rapidement les configurations légitimes. Une autre perspective intéressante porte sur la prise en compte des symétries pour la détection des cycle divergents. Ces perspectives ouvrent la voie à une meilleure compréhension des dynamiques de synchronisation dans les systèmes distribués complexes, et à la mise au point de nouvelles méthodes plus efficaces et optimisées pour traiter un éventail plus large de cas d’applications.

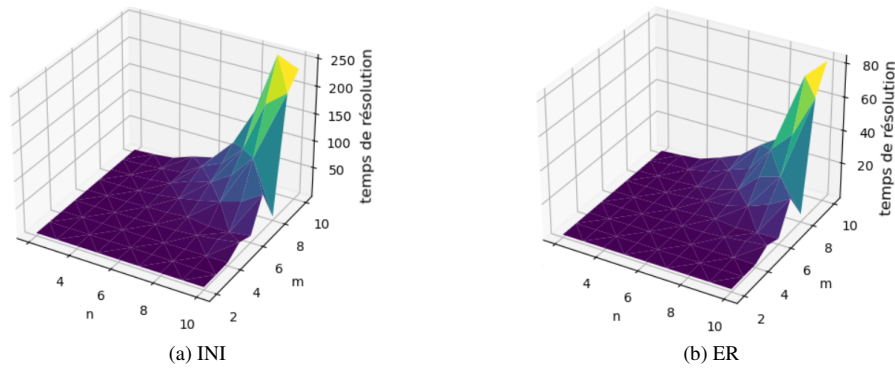


FIGURE 5 – Temps de résolution en secondes pour la divergence en fonction du nombre de nœuds n et de la période m pour le graphe des anneaux avec le modèle initial (INI) et augmenté par l'élimination des rotations (ER)

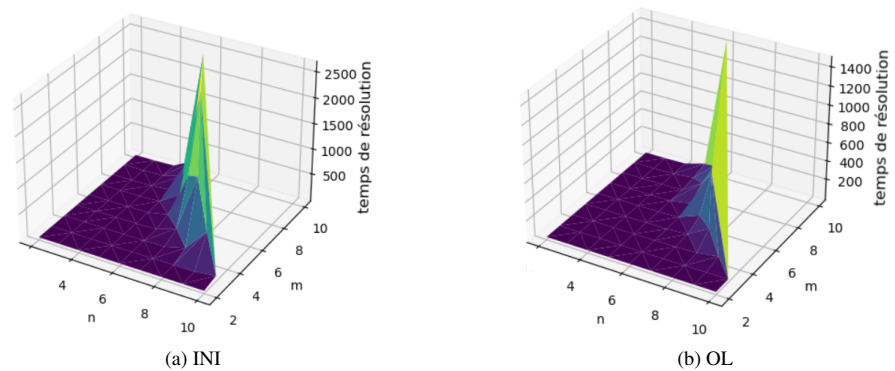


FIGURE 6 – Temps de résolution en secondes pour la divergence en fonction du nombre de nœuds n et de la période m pour les graphes en étoile avec le modèle initial (INI) et augmenté par l'ordre lexicographique (OL)

Remerciements

Ce travail est partiellement soutenu par le projet ANR-24-CE23-6126 (BforSAT) financé par l'agence nationale de recherche. Il a bénéficié d'un accès aux ressources HPC de la « Plateforme MatriCS » de l'Université de Picardie Jules Verne, qui est cofinancée par l'Union Européenne avec le Fonds Européen de Développement Régional (FEDER) et le Conseil Régional des Hauts-De-France entre autres.

Références

- [1] Karine Altisen, Stéphane Devismes, Swan Dubois, and Franck Petit. *Introduction to Distributed Self-Stabilizing Algorithms*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2019.
- [2] Anish Arora, Shlomi Dolev, and Mohamed G. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1 :11–18, 1991.
- [3] Armin Biere. Handbook of satisfiability. In *Frontiers in Artificial Intelligence and Applications*, pages 75–98. IOS Press, 2009.
- [4] Armin Biere et al. Symbolic model checking without bdds. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 193–207, 1999.
- [5] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froyleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *International Conference on Computer Aided Verification, CAV*, volume 14681 of *LNCS*, pages 133–152. Springer, 2024.
- [6] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of Satisfiability : Second Edition*. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021.
- [7] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of the Workshop on Logic of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1982.
- [8] Stephen A. Cook. The complexity of theorem-proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.
- [9] Jean-Michel Couvreur, Nissim Francez, and Mohamed G. Gouda. Asynchronous unison (extended abstract). In *The 12th International Conference on Distributed Computing Systems (ICDCS)*, pages 486–493. IEEE Computer Society, 1992.
- [10] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11) :643–644, 1974.
- [11] Danny Dolev and Dahlia Malkhi. Consensus : Perspectives and challenges. In *Proceedings of the Tenth International Workshop on Distributed Algorithms (WDAG)*, pages 1–12, 1995.
- [12] João P. Marques-Silva and Karem A. Sakallah. Grasp : A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5) :506–521, 1999.

Satisfiabilité pour le décodage par syndrome

Carl Berton, Sami Cherif, Claire Delaplace

Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

carl.berton@etud.u-picardie.fr, {sami.cherif;claire.delaplace}@u-picardie.fr

Résumé

Le problème du décodage par syndrome d'un code linéaire binaire consiste à trouver une solution de faible poids de Hamming pour un système linéaire bruité défini sur le corps fini à deux éléments. Dans cet article, nous explorons plusieurs modélisations de ce problème basées sur la satisfiabilité. La première repose sur une représentation XNF, qui combine des contraintes clausales utilisant à la fois l'opérateur de disjonction classique (OR) et l'opérateur de disjonction exclusive (XOR). Ce modèle est exploité par des solveurs de satisfiabilité dédiés aux problèmes cryptographiques. Un second modèle proposé repose uniquement sur des contraintes clausales classiques, générant ainsi des formules CNF qui pourraient être résolues à l'aide des solveurs SAT de l'état de l'art. Nous comparons ces approches afin d'évaluer leur efficacité dans la résolution du problème du décodage par syndrome.

Mots-clés

Décodage par syndrome, Satisfiabilité, Cryptographie

Abstract

The syndrome decoding problem for a linear binary code consists in finding a low Hamming weight solution to a noisy linear system defined over the finite field of two elements. In this paper, we explore several SAT-based models for solving this problem. The first approach relies on an XNF representation, which combines clausal constraints using both classical disjunction (OR) and exclusive disjunction (XOR). This model is designed for SAT solvers specialized in cryptographic problems. The second proposed model relies solely on classical clausal constraints, generating CNF formulas that can be solved using state-of-the-art SAT solvers. We compare these approaches to assess their efficiency in solving the syndrome decoding problem.

Keywords

Syndrome Decoding, Satisfiability, Cryptography

1 Introduction

Le problème de satisfiabilité propositionnelle (SAT) consiste à déterminer, étant donné une formule en Forme Normale Conjonctive (CNF), s'il existe une affectation des variables qui la satisfait [8]. SAT est un problème de décision fondamental en informatique et en intelligence artificielle, utilisé pour résoudre une large gamme de problèmes

dans divers domaines, tels que la cryptographie [17, 26, 28], la vérification matérielle et logicielle [15], la planification [23], et bien d'autres. Il a été le premier problème à avoir été démontré comme étant NP-complet [9]. Certains problèmes, notamment en cryptanalyse, sont naturellement formulés sous une autre forme appelée forme normale algébrique (ANF). Bien qu'une conversion en forme normale conjonctive (CNF) soit possible, elle peut provoquer une explosion combinatoire du nombre de variables et de clauses [16], compromettant ainsi l'efficacité de la résolution. Pour pallier cette limitation, certains solveurs, tels que CryptoMiniSat [27], utilisent une structure hybride de formules, combinant à la fois des clauses CNF classiques et des clauses exprimées à l'aide de l'opérateur de disjonction exclusive (XOR), un format connu sous le nom de XNF (XOR-CNF) [22].

Dans ce contexte, nous nous intéressons aux problèmes de décodage. En effet, pour assurer la transmission fiable d'informations sur des canaux bruités, on utilise des codes correcteurs, notamment des codes linéaires binaires, qui ajoutent une redondance pour détecter et corriger les erreurs [24]. Les codes utilisés en théorie de l'information sont construits de manière à ce que ce problème soit facile à résoudre, afin de garder une communication fluide. Cependant, décoder un code linéaire binaire dans le cas général est un problème NP-difficile [3]. Les cryptologues se sont intéressés à ce problème dès les années 70 [19], quand McEliece a mis en place un premier schéma de chiffrement à clé publique dont la sécurité repose sur la difficulté de décoder un code linéaire binaire. Avec l'avènement de la cryptographie post-quantique, la cryptographie à base de code s'est beaucoup développée au cours des dernières années avec de nombreux efforts à la fois du point de vue des constructions et des attaques [20, 11, 21].

Un problème qui intéresse particulièrement les cryptologues est celui du décodage par syndrome qui consiste à retrouver le bruit à partir de la matrice de parité du code [4]. Concrètement, ce problème revient à résoudre un système linéaire sous-déterminé modulo 2 avec la contrainte que la solution doit être de poids de Hamming faible. Bien que ce problème semble se modéliser facilement sous forme de contrainte XNF, il n'existe à notre connaissance aucuns travaux qui visent à résoudre le problème du décodage par syndrome en utilisant des solveurs SAT. À noter que d'autres problèmes similaires, tel que le décodage par maximum de vraisemblance, sont connus pour se modéliser sous

forme de problèmes de programmation linéaire en nombres entiers (PLNE) [12]. Il faut toutefois remarquer que ces travaux se focalisent sur des codes particuliers, à savoir des codes type turbo [5] et des codes de parité à faible densité (LDPC) [13], pour lesquels le décodage est plus facile que dans le cas aléatoire qui nous intéresse dans ce papier.

Dans cet article, nous présentons des travaux préliminaires visant à étudier deux approches de modélisation pour le problème de décodage par syndrome. La première repose sur une représentation XNF, qui permet d'exprimer directement les contraintes de parité, complétée par des contraintes clausales classiques pour intégrer la contrainte de poids sur le vecteur d'erreur. La seconde approche consiste à formuler le problème sous un format CNF classique, en proposant deux modélisations alternatives où les contraintes XOR sont réécrites à l'aide de contraintes Pseudo-Booléennes (PB). Cette dernière approche présente l'avantage d'intégrer plus naturellement la spécification du poids d'erreur lors de la reformulation des XOR et donc de réduire la combinatoire de l'espace de recherche. Il est largement admis que les représentations clausales classiques sont moins efficaces que les représentations XNF pour les problèmes cryptographiques. Notre objectif est donc de comparer une modélisation XNF standard à une approche CNF enrichie par des contraintes Pseudo-Booléennes, afin d'évaluer leur performances respectives et d'identifier si une représentation CNF classique peut rester pertinente pour certains types de problèmes cryptographiques comme le décodage par syndrome.

Cet article est organisé comme suit. Nous introduisons dans la section 2 les notions fondamentales nécessaires à notre étude, en rappelant les différentes représentations clausales utilisées, avant de présenter en détail le problème du décodage par syndrome. Ensuite, nous poursuivons dans la section 3 avec la modélisation de ce problème à la fois en format XNF et CNF classique. Nous analysons les résultats expérimentaux obtenus afin de comparer ces différentes approches dans la section 4. Enfin, on conclut et discute des travaux futurs dans la section 5.

2 Préliminaires

2.1 Satisfiabilité

Soit $X = \{x_1, x_2, \dots, x_n\}$ un ensemble de variables booléennes qui peuvent prendre les valeurs *Vrai* ($\top$ ou 1) ou *Faux* ($\perp$ ou 0). Un littéral est soit une variable $x \in X$, soit sa négation $\bar{x}$. Une formule CNF ϕ est une conjonction de clauses $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_m$, où chaque clause C_j est une disjonction ($\vee$) de littéraux. Une affectation $\alpha : X \rightarrow \{\text{True}, \text{False}\}$ est une fonction associant à chaque variable de X une valeur de vérité. Ainsi, une clause est satisfaite par une interprétation α si au moins l'un de ses littéraux est satisfait par α . Une formule CNF ϕ est satisfaite par une interprétation α si toutes ses clauses sont satisfaites par α . Dans ce cas, on dit que la formule est satisfiable et que α est un modèle de ϕ . Le problème de satisfiabilité propositionnelle (SAT) consiste à déterminer si une formule CNF donnée est satisfiable. Les solveurs SAT

modernes sont extrêmement performants et parviennent à résoudre des formules comportant un grand nombre de variables et de clauses en un temps remarquablement court, malgré la difficulté théorique établie du problème. En effet, ces solveurs sont basés sur l'algorithme CDCL [25] (Conflict-Driven Clause Learning) et emploient donc des mécanismes puissants comme l'analyse de conflits, les redémarrages ou encore des heuristiques de branchement dédiées [8].

Une autre forme de représentation particulièrement utilisée pour les problèmes cryptographiques dans le cadre de la satisfiabilité est le format XNF (XOR-CNF). Contrairement aux formules CNF classiques, qui utilisent uniquement des contraintes clausales avec des opérations de disjonction logique, les XNF emploient en plus des clauses basées sur l'opérateur XOR (disjonction exclusive). Une formule XNF est donc une conjonction de clauses CNF et XOR, où chaque clause XOR est une disjonction exclusive de littéraux. Une clause XOR s'écrit donc sous la forme $C = l_1 \oplus l_2 \oplus \dots \oplus l_k$ où chaque l_i est un littéral, et $\oplus$ représente l'opérateur XOR. Il est possible d'ajouter la constante $\top$ à une clause pour indiquer un littéral qui est toujours satisfait. Contrairement à l'opération OR, une clause XOR est satisfaite par une interprétation α si et seulement si un nombre impair de ses littéraux sont satisfaits. Une formule XNF est donc satisfaite par α si toutes ses clauses classiques et XOR sont satisfaites.

Dans nos modèles, nous utiliserons également des contraintes Pseudo-Booléennes (PB) qui imposent une borne sur une somme pondérée de littéraux comme suit :

$$\left(\sum_{i=1}^h a_i \times l_i \right) \circ k$$

où $a_i, k \in \mathbb{N}$ et $\circ \in \{\leq, =, \geq\}$. Dans le cas où tous les poids sont fixés à 1 (et peuvent donc être omis), de telles contraintes sont plus couramment appelées contraintes de cardinalité. Les contraintes PB et de cardinalité sont typiquement utilisées dans les solveurs pour imposer des bornes pertinentes lors de la recherche et peuvent être encodées efficacement sous forme clausale [6].

2.2 Le Problème du Décodage par Syndrome

En communications numériques, la transmission fiable d'informations est essentielle. Les canaux de transmission sont souvent bruités, ce qui engendre des erreurs dans les messages reçus. Pour y remédier, on utilise des codes correcteurs, notamment des codes linéaires binaires, qui intègrent une redondance dans la transmission [24].

Soit $\mathbb{F}_2$ le corps fini à deux éléments et soient n, k , et t des entiers tels que $0 < k < n$ et $0 < t < \frac{n}{2}$. Pour tout vecteur $v \in \mathbb{F}_2^n$, on note $\text{wt}(v)$ son poids de Hamming, c'est-à-dire le nombre de coefficients non nuls de v . Étant donné une matrice H de taille $(n - k) \times n$ à coefficients dans $\mathbb{F}_2$, le code linéaire binaire C de matrice de parité H est défini par l'ensemble des vecteurs v appartenant au noyau à droite de H . Autrement dit,

$$C = \{v \in \mathbb{F}_2^n : Hv = 0\} \quad (1)$$

Soit $u \in \mathbb{F}_2^n$, u peut s'exprimer sous la forme :

$$u = v + e$$

où $v \in C$ est un mot de code et e est un vecteur d'erreur, qu'on suppose généralement suffisamment petit pour que le décodage fonctionne. Typiquement, si on se place dans le cadre d'un décodage à distance complète, on a $wt(e) \leq d$ avec :

$$d = \min\{wt(u + v) : (u, v) \in C^2, u \neq v\}$$

On appelle syndrome de u le vecteur $s \in \mathbb{F}_2^{(n-k)}$ tel que

$$Hu = s$$

Comme $v \in C$, on a $Hu = H(v + e) = Hv + He = He$. Autrement dit, le syndrome de u est identique à celui de e . Le décodage par syndrome consiste à décoder u en effectuant les étapes suivantes :

1. Calculer le syndrome $s = Hu$.
2. Résoudre le problème de décodage par syndrome, c'est à dire trouver un vecteur $e \in \mathbb{F}_2^n$ tel que $He = s$ et $wt(e) \leq t$.
3. Récupérer le mot de code corrigé $v = u + e$.

Il est clair que la difficulté de ce problème se trouve à l'étape 2. Ici, il est demandé de trouver une solution de poids de Hamming faible à un système linéaire sous-déterminé. Il existe des codes particulier pour lesquels ce problème est facile à résoudre, mais dans le cas général, il est NP-difficile. Formellement nous définissons le problème de la manière suivante :

Définition 1 (décodage par syndrome à distance complète)

Étant donné un code linéaire binaire C de longueur n , de dimension k et de distance minimale d représenté par sa matrice de parité $H \in \mathbb{F}_2^{(n-k) \times n}$, un paramètre $t < d$ et un vecteur $s \in \mathbb{F}_2^{n-k}$, résoudre le problème du décodage par syndrome consiste à retrouver $e \in \mathbb{F}_2^n$ tel que $He = s$ et $wt(e) \leq t$.

Si on considère plutôt des représentants sur $\mathbb{Z}$ des éléments de $\mathbb{F}_2$, un système de décodage par Syndrome, qu'on appellera $Syndrome(k, n, t)$, peut s'écrire comme suit :

$$He = s \pmod{2} \quad (2a)$$

$$\sum_{i=0}^n e_i \leq t, \quad (2b)$$

où H est une matrice de $n - k$ lignes et n colonnes à coefficients entiers, et e et s sont des vecteurs d'entiers respectivement de n et $n - k$ coefficients. On note $H_{i,j}$ le coefficient sur la i -ème ligne, j -ème colonne de H . Plus précisément, $Syndrome(k, n, t)$ est défini comme suit :

$$E_i : \sum_{\substack{0 \leq j \leq n-1 \\ H_{i,j}=1}} e_j = s_i \pmod{2} \quad \forall 0 \leq i \leq n - k - 1 \quad (3a)$$

$$\sum_{i=0}^{n-1} e_i \leq t \quad (3b)$$

Les cryptologues s'intéressent en particulier au cas des codes linéaires binaires aléatoires. Ces codes sont connus pour avoir une distance minimale qui atteint la borne de Gilbert-Varshamov, notée d_{GV} et qui satisfait

$$\frac{k}{n} = 1 - \mathcal{H}\left(\frac{d_{GV}}{n}\right),$$

où $\mathcal{H}$ est la fonction d'entropie binaire :

$$H : [0, 1] \rightarrow [0, 1] \\ x \mapsto -x \log_2(x) - (1 - x) \log_2(1 - x).$$

Si de plus le débit du code k/n est proche de $1/2$, alors cette instance du problème (1) est difficile.

3 Modélisation du problème de décodage par syndrome

Dans cette section, on s'intéresse à la modélisation du problème de décodage par syndrome grâce à la satisfiabilité propositionnelle. En particulier on introduit des modélisations basées à la fois sur les formats XNF et CNF.

3.1 Modélisation XNF

Le système de décodage par syndrome peut être modélisé en format XNF en utilisant l'opérateur XOR pour vérifier la parité dans les équations E_i où $0 \leq i \leq n - k - 1$. Pour simplifier les notations, on dénote $m = n - k - 1$ et V_{E_i} l'ensemble des indices des variables présentes dans l'équation E_i . Plus formellement, pour tout $i \in \{0, \dots, m\}$, on a $V_{E_i} = \{j \mid 0 \leq j \leq n - 1 \text{ et } H_{i,j} = 1\}$. Le système $Syndrome(k, n, t)$ peut donc s'écrire sous la forme suivante :

$$\left(\bigoplus_{j \in V_{E_i}} e_j \right) \bigoplus_{s_i=0} \top \quad \forall 0 \leq i \leq m \quad (4a)$$

$$\sum_{j=0}^{n-1} e_j \leq t \quad (4b)$$

On peut également remarquer que lorsque la taille d'une clause XOR est supérieure au poids t , le solveur doit se fier à la contrainte (4b) impliquant des variables qui pourrait apparaître dans d'autres clauses XOR pour éliminer des sous-espaces de la recherche où il ne peut pas y avoir de solution. Pour l'aider à raisonner directement sur les variables d'une seule clause, on peut donc rajouter les contraintes redondantes suivantes :

$$\sum_{j \in V_{E_i}} e_j \leq t \quad \forall 0 \leq i \leq m \text{ tel que } |V_{E_i}| > t \quad (5)$$

3.2 Modélisation CNF

Il est largement admis que les représentations clausales (avec disjonction classique) sont moins efficaces sur les problèmes cryptographiques que les représentations XNF,

où on admet des clauses avec des disjonctions exclusives. En effet, une conversion classique d'une clause XOR en format CNF sans ajout de variables additionnelles peut avoir un coût exponentiel [16]. Dans notre cas, on veut introduire des modèles CNF dédiés au problème du décodage par Syndrome et qui seraient compétitifs.

Pour cela, on définit, pour $0 \leq i \leq m$, l'ensemble suivant :

$$K_{E_i} = \{0 \leq j \leq \min(|V_{E_i}|, t) \mid j \bmod 2 = s_i\}$$

Clairement, ces ensembles représentent les valeurs possibles que peuvent prendre les sommes $\sum_{j \in V_{E_i}} e_j$ dans une équation E_i donnée selon la parité indiquée par s_i . Il est important de noter que ces ensembles prennent naturellement en compte la valeur du poids t pour éliminer les sommes qui ne produisent pas une solution faisable. Ainsi, pour représenter le choix des valeurs possibles dans ces sommes dans les différentes équations, on introduit les variables $x_{i,v}$ pour tout $(i, v) \in \{0, \dots, m\} \times K_{E_i}$, qui sont affectés à vrai si et seulement si la somme des variables dans l'équation E_i est égale à v . Plus formellement, $x_{i,v}$ est affectée à vrai si et seulement si $\sum_{j \in V_{E_i}} e_j = v$. Le système $Syndrome(k, n, t)$ peut donc se réécrire comme suit :

$$\sum_{j \in V_{E_i}} e_j = \sum_{v \in K_{E_i} \setminus \{0\}} v \cdot x_{i,v} \quad \forall 0 \leq i \leq m \quad (6a)$$

$$\sum_{v \in K_{E_i}} x_{i,v} = 1 \quad \forall 0 \leq i \leq m \quad (6b)$$

$$\sum_{i=0}^{n-1} e_i \leq t \quad (6c)$$

La contrainte (6a) assure que la somme des variables dans une équation est égale à l'une des valeurs possibles selon sa parité tandis que la contrainte (6b) est nécessaire pour s'assurer qu'une seule et unique valeur est sélectionnée à la fois pour une équation donnée. On peut réécrire (6a) sous forme de contrainte Pseudo-Booléenne (PB) comme suit :

$$\sum_{j \in V_{E_i}} e_j + \sum_{v \in K_{E_i} \setminus \{0\}} v \cdot \bar{x}_{i,v} = \sum_{v \in K_{E_i}} v \quad \forall 0 \leq i \leq m \quad (6a')$$

La représentation ainsi définie peut s'apparenter à la représentation par polyèdres introduite dans [10, 29], mais en rajoutant moins de variables et en prenant en compte naturellement le poids induit par le problème de décodage par syndrome dans chaque conversion de clause XOR.

On peut également remarquer que les éléments de l'ensemble K_{E_i} peuvent être déterminés par incréments de 2, vu qu'on raisonne sur la parité. Pour tirer parti de cette caractéristique, on peut adapter le sens sémantique des variables $x_{i,v}$ pour représenter le fait que la somme des variables dans l'équation E_i est inférieure ou égale à v . Plus formellement, $x_{i,v}$ est affectée à vrai si et seulement si

$\sum_{j \in V_{E_i}} e_j \leq v$. Cela nous permet donc de reformuler le système syndrome comme suit :

$$\sum_{j \in V_{E_i}} e_j = \sum_{v \in K_{E_i} \setminus \{0,1\}} 2 \cdot x_{i,v} + s_i \quad \forall 0 \leq i \leq m \quad (7a)$$

$$\bar{x}_{i,v} \vee x_{i,v-2} \quad \forall 0 \leq i \leq m, v \in K_{E_i} \setminus \{0, 1, 2, 3\} \quad (7b)$$

$$\sum_{i=0}^{n-1} e_i \leq t \quad (7c)$$

La contrainte (7a) constitue désormais une représentation plus naturelle de la parité des sommes, permettant une meilleure intégration des contraintes de parité dans la modélisation CNF du problème de décodage par syndrome. On peut également la réécrire naturellement sous forme de contrainte PB comme indiquée ci-dessous, où $\max$ désigne l'opérateur usuel renvoyant l'élément maximal dans un ensemble donné. Quant à la contrainte (7b), elle assure que les valeurs possibles des sommes sont bien sélectionnées de manière cohérente et ordonnée, c'est à dire $x_{i,v} \rightarrow x_{i,v-2}$, en respectant l'incrément par 2 imposée par la structure des ensembles K_{E_i} .

$$\sum_{j \in V_{E_i}} e_j + \sum_{v \in K_{E_i} \setminus \{0,1\}} 2 \cdot \bar{x}_{i,v} = \max K_{E_i} \quad \forall 0 \leq i \leq m \quad (7a')$$

Cette représentation se rapproche ainsi davantage de celle proposée dans [14] pour la certification des raisonnements de parité. A noter que la deuxième représentation est non seulement plus naturelle mais elle permet également de réduire la complexité de l'encodage. En effet, la complexité de l'encodage d'une contrainte PB est souvent régie par le nombre de littéraux à sommer et le poids maximal des littéraux. Dans notre cas, le nombre de littéraux à sommer reste de même ordre dans les deux modèles. Plus formellement, pour $0 \leq i \leq m$, on peut remarquer que $|K_{E_i}| \leq |V_{E_i}| \leq n$ où n désigne la longueur du code et ainsi le nombre de littéraux sommés est de l'ordre de $O(n)$ dans les deux modèles CNF. Cependant, dans le premier modèle CNF, le poids maximal est $\max K_{E_i}$, pouvant atteindre n , tandis que le poids maximal du deuxième modèle est toujours fixe et égal à 2.

4 Évaluation Expérimentale

4.1 Protocole expérimental

Pour évaluer les modèles introduits dans la section précédente, nous avons testé des instances issues du Decoding Challenge¹, avec des valeurs de n comprises entre 10 et 150. Nous notons nos modèles (4), (4-5), (6) et (7) respectivement XNF1, XNF2, CNF1 et CNF2. Les expérimentations ont été réalisées avec les solveurs CryptoMiniSat [27] pour les instances XNF et CaDiCaL [7] pour les

1. <https://decodingchallenge.org/syndrome>

Instance		CardNetwork				SortNetwork			
n	t	XNF1	XNF2	CNF1	CNF2	XNF1	XNF2	CNF1	CNF2
10	4	163.98	191.47	78.47	78.23	197.40	162.38	61.84	72.94
20	5	91.83	61.58	1.41	7.50	120.77	86.90	1.54	2.59
30	7	12.39	12.46	43.93	26.00	31.74	23.98	35.16	35.19
40	8	47.66	48.67	8.41	42.98	99.89	65.42	16.58	27.84
50	9	88.63	95.39	38.32	24.10	77.11	120.20	26.18	31.54
60	10	123.83	111.76	31.16	48.41	69.28	92.92	33.62	38.30
70	11	137.69	135.75	81.34	96.33	220.39	195.08	126.20	43.13
80	12	1831.61	2314.17	752.01	368.60	7416.13	1038.27	421.41	542.20
90	13	10788.26 (17)	6859.91	3636.34	2623.91	8000.81 (8)	5556.04	7321.13	1711.03
100	14	16510.45 (11)	21601.34 (18)	11696.80 (15)	7479.82 (17)	2977.08 (2)	18901.42 (18)	9276.91 (13)	13002.32 (14)
110	16	13202.29 (9)	15629.83 (16)	13770.29 (18)	10099.52 (18)	1853.72 (5)	12808.43 (12)	12199.81 (17)	13870.86 (18)
120	17	392.69 (3)	23613.99 (16)	8323.67 (12)	7184.07 (13)	6006.44 (5)	20028.54 (14)	13394.07 (12)	6804.34 (9)
130	18	7118.13 (4)	13741.32 (8)	3962.57 (3)	2882.16 (4)	4403.01 (3)	9863.13 (6)	7890.08 (5)	1141.45 (1)
140	19	- (0)	9196.25 (4)	8583.47 (4)	- (0)	- (0)	3203.19 (1)	1480.40 (2)	3245.78 (3)
150	20	71.35 (1)	1536.56 (1)	2304.49 (2)	271.72 (1)	- (0)	- (0)	- (0)	- (0)
SOMME		50580.77 (205)	95150.45 (243)	53312.68 (234)	31233.35 (233)	31473.78 (183)	72145.90 (231)	52284.92 (229)	40569.51 (225)

TABLE 1 – Somme des temps de résolution en secondes pour les instances résolues pour chaque taille n et t selon les différentes modélisations et encodages du problème de décodage par syndrome. Si des problèmes n’ont pas été résolus dans la limite du temps alloué, le nombre d’instances résolues est indiqué entre ‘()’. Les meilleurs résultats sont marqués en gras et soulignés, d’abord en fonction du nombre d’instances résolues, puis, en cas d’égalité, en fonction du temps de résolution.

instances CNF. Tous les tests ont été exécutés sur la plateforme de calcul MatriCS² sur la partition bigmem, équipée de 12 serveurs bi-processeur Intel Xeon E5-2680 v4 (2.40 GHz), chacun doté de 512 Go de mémoire, avec une limite de temps fixée à 3600 secondes (soit 1 heure) par instance. Nous avons utilisé la librairie PySAT³ pour encoder nos modèles ainsi que les contraintes pseudo-bouliennes. En particulier, nous avons testé deux encodages de la littérature pour les contraintes de cardinalité présentes dans tous les modèles : SortNetwork [2] et CardNetwork [1]. Pour une contrainte de cardinalité donnée sous la forme $\sum_{i=1}^h l_i \leq b$, la complexité en termes de nombres de clauses (et variables) générées est de $O(h \log^2 h)$ pour SortNetwork et de $O(h \log^2 b)$ pour CardNetwork. Ainsi, la complexité pour notre modèle XNF1 est respectivement de l’ordre de $O(n \log^2 n)$ et de $O(n \log^2 t)$ pour ces encodages (de même pour XNF2 avec un facteur n additionnel). Pour les modèles CNF, nous avons également sélectionné l’encodage BinMerge [18] pour encoder les contraintes pseudo-bouliennes (spécifiquement les contraintes (6a’) et (7a’)), dont la complexité est de l’ordre de $O(h \log^2 h \log W)$ où h désigne le nombre de littéraux dans la somme et W le poids maximal. Ainsi, la complexité de nos modèles CNF1 et CNF2 est respectivement de l’ordre de $O(n \log^3 n)$ et $O(n \log^2 n)$. Enfin, pour chaque valeur de n , chaque modèle et chaque encodage choisi pour les contraintes de cardinalité, nous avons généré 20 instances avec des graines aléatoires distinctes, totalisant ainsi 2400 instances du problème de décodage par syndrome.

4.2 Résultats globaux

Les résultats en termes de nombre d’instances résolues et de temps de résolution sont présentés dans le tableau 1. Pour l’encodage CardNetwork, XNF2 parvient à résoudre le plus

grand nombre d’instances (243), avec une légère avance sur CNF1 et CNF2, qui en résolvent respectivement 234 et 233, soit 9 et 10 de moins. Cependant, ces deux modélisations CNF se démarquent par leur meilleure efficacité en termes de temps de résolution, avec des gains d’environ 44% pour CNF1 et 67% pour CNF2 par rapport à XNF2. La modélisation XNF1 restent nettement moins performantes, avec seulement 205 instances résolues. On note également que CNF2 est environ 38% plus rapide que XNF1, tout en résolvant 28 instances supplémentaires. Concernant l’encodage SortNetwork, XNF2 reste la modélisation la plus performante en termes d’instances résolues (231), suivie de CNF1 (229) puis CNF2 (225), avec respectivement un écart de 2 et 6 instances par rapport à XNF2. Toutefois, les performances en termes de temps de résolution sont à l’avantage des modélisations CNF, avec un gain d’environ 28% pour CNF1 et 44% pour CNF2 sur la somme totale des temps de résolution par rapport à XNF2. Comme avec CardNetwork, XNF1 reste en retrait, ne résolvant que 183 instances, soit 42 de moins que CNF2.

De manière générale, les modélisations sont plus performantes en nombre d’instances résolues lorsqu’elles sont associées à l’encodage CardNetwork. Par exemple, XNF2 résout 12 instances de plus avec cet encodage comparé à SortNetwork. Cela peut s’expliquer par le fait que CardNetwork génère moins de variables et de clauses, ce qui rend les instances plus faciles à résoudre. Globalement, CNF2 avec CardNetwork apparaît comme un excellent compromis entre nombre d’instances résolues et temps de résolution, affichant le meilleur temps de résolution total, tout en résolvant 50 instances de plus que XNF1 avec SortNetwork, pourtant deuxième en termes de temps. Enfin, CNF2 avec CardNetwork est environ 41% plus rapide que CNF1 avec le même encodage, tout en ne résolvant qu’une seule instance de moins. Nous remarquons également que XNF2 avec CardNetwork semble être la modélisation la plus ro-

2. <https://www.matrics.u-picardie.fr/>

3. <https://pysathq.github.io/>

Instance n	CardNetwork				SortNetwork			
	XNF1	XNF2	CNF1	CNF2	XNF1	XNF2	CNF1	CNF2
10-50	404.49	409.57	170.55	178.81	526.92	458.87	141.29	170.11
60-100	72591.84	38222.93	34197.65	21417.07	126683.7	32983.73	42379.28	36936.97
110-150	319584.45	261717.95	256544.49	250837.47	325463.16	287103.3	265364.36	273462.43
TOTAL	392580.77	300350.45	290912.68	272433.35	452673.78	320545.9	307884.92	310569.51

TABLE 2 – Score PAR1 obtenu pour chaque famille d’instances selon les différentes modélisations et encodages du problème de décodage par syndrome. Les meilleurs résultats sont marqués en gras et soulignés.

buste en termes de nombre d’instances résolues, notamment lorsque la difficulté augmente. Elle fait la différence sur les modélisations CNF pour les tailles $n=120$, 130 et 140 , avec respectivement 16, 8 et 4 instances résolues contre 12, 3 et 4 pour CNF1, et 13, 4 et 0 pour CNF2 avec le même encodage.

Du point de vue des instances, nous avons classé notre benchmark en trois catégories : les instances faciles (n de 10 à 50), intermédiaires (n de 60 à 100) et difficiles (n de 110 à 150). Pour chaque famille d’instances, le PAR1 a été calculé en cumulant les temps de résolution et en indiquant la limite de temps pour les instances non résolues. Ces résultats sont présentés dans le tableau 2.

Pour les instances faciles, toutes les modélisations ont permis de résoudre l’ensemble des instances. Quelle que soit l’encodage utilisé, les modélisations CNF surpassent nettement les modélisations XNF. CNF1 avec SortNetwork affiche le meilleur temps, avec un gain d’environ 17% par rapport à CNF2 avec le même encodage (deuxième meilleur temps), et un gain d’environ 65% par rapport à XNF1 avec CardNetwork, qui reste la plus performante parmi les modélisations XNF. Pour les instances intermédiaires, CNF2 associée à CardNetwork surpasse les autres modélisations, avec un gain d’environ 35% par rapport à XNF2 associée à SortNetwork, qui se classe en deuxième position. Bien que CNF2 résolve une instance de moins que XNF2 dans cette catégorie, elle reste significativement plus rapide sur celles qu’elle parvient à traiter. Enfin, en ce qui concerne les instances difficiles, CNF2 avec CardNetwork conserve la première place en termes de score PAR1, suivie de près par CNF1, puis XNF2 avec le même encodage. Même si XNF2 parvient à résoudre davantage d’instances que les deux modélisations CNF, respectivement 6 et 9 instances de plus que CNF1 et CNF2, ces dernières restent plus performantes en termes de temps de résolution sur les instances qu’elles traitent.

Les résultats globaux montrent que les modélisations couplées à l’encodage CardNetwork offrent les meilleures performances. CNF2 avec CardNetwork affiche le score PAR1 le plus bas avec un gain d’environ 6% par rapport à CNF1 et de 9% par rapport à XNF2, tandis que XNF1 associée au même encodage et toutes les modélisations utilisant SortNetwork restent en retrait.

4.3 CNF vs XNF

Dans cette section, nous analysons plus en détails les différences de performances entre les modélisations CNF et

XNF. La Figure 1 présente l’évolution du temps moyen de résolution (en secondes) en fonction de la taille des instances (n), pour les différentes modélisations encodées avec CardNetwork. La limite de temps est prise en compte dans le calcul de la moyenne lorsqu’une instance n’est pas résolue dans le temps imparti..

On observe globalement une croissance très rapide du temps moyen avec la taille des instances. Cette croissance devient particulièrement marquée à partir de $n = 80$, avec des augmentations de $n = 70$ à $n = 80$ d’environ 1231% et 1604% pour XNF1 et XNF2, 824% et 282% pour CNF1 et CNF2. Ces valeurs illustrent l’explosion des temps de calcul, notamment pour les modélisations XNF. Pour les tailles d’instance allant de $n = 10$ à $n = 120$, on observe que les temps moyens des représentations CNF restent systématiquement inférieurs à ceux des XNF, à l’exception de $n = 30$ où la tendance s’inverse. De plus, on observe que XNF1 est nettement moins performant que XNF2, particulièrement à partir de $n = 90$ jusqu’à $n = 120$. On remarque également que CNF2 est constamment en dessous de CNF1 pour $n = 80$ à $n = 120$. Au-delà de $n = 120$, il devient

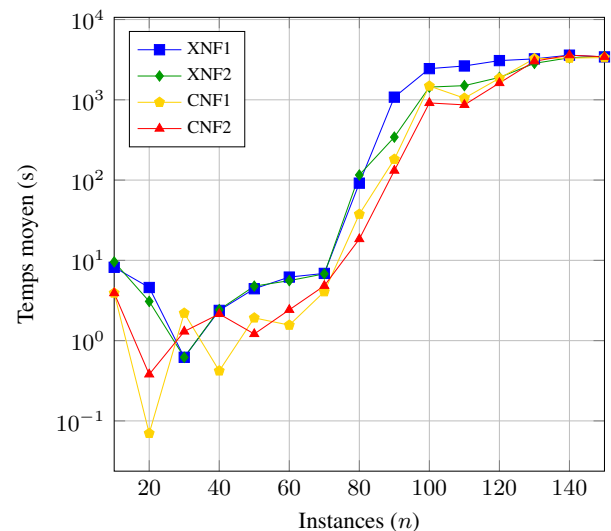


FIGURE 1 – Temps moyens en secondes des différentes modélisations encodées avec CardNetwork du problème de décodage par syndrome. Pour les problèmes non résolus dans le temps imparti, la limite de temps est prise en compte dans la moyenne.

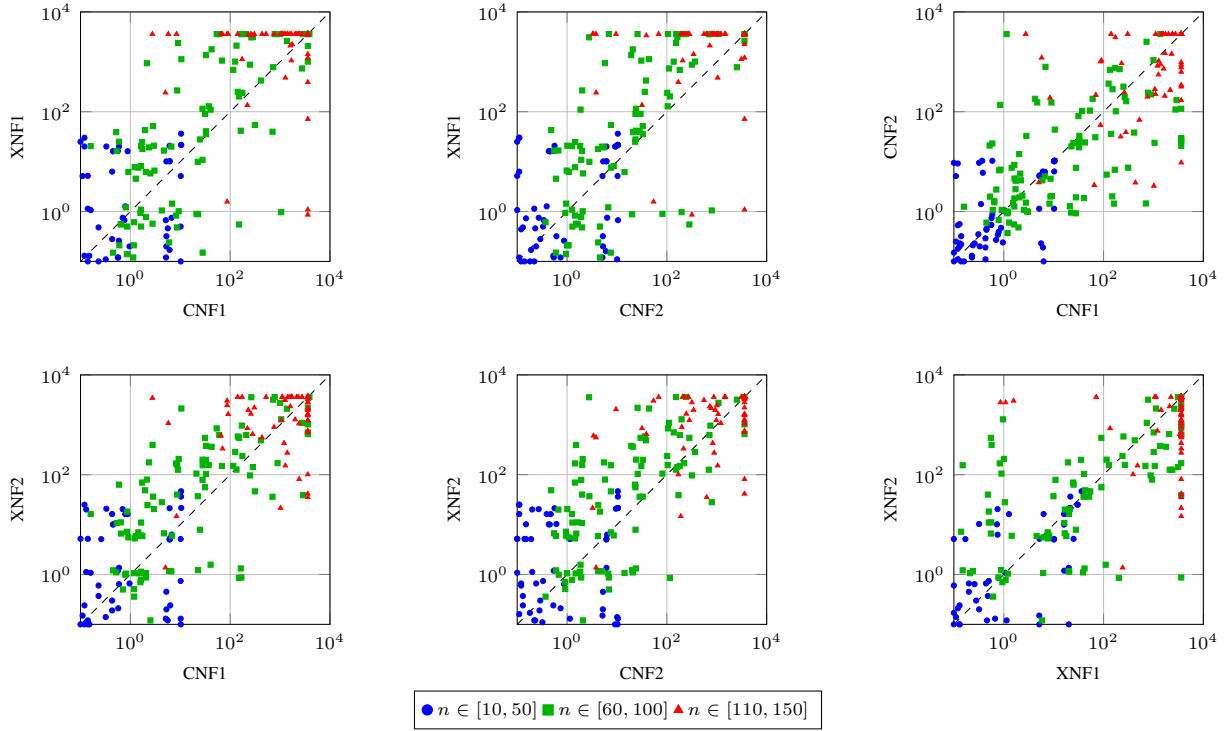


FIGURE 2 – Comparaison des temps de résolution en secondes par instance pour les différentes modélisations avec l’encodage CardNetwork. La limite de temps est prise en compte pour les instances non résolus dans le temps imparti.

plus difficile d’interpréter les courbes, vu la difficulté accrue du problème pour ces valeurs, faisant converger toutes les courbes vers la limite de temps fixée.

La Figure 2 illustre une comparaison des temps d’exécution par instance entre les différentes modélisations encodées avec CardNetwork. Parmi les modèles XNF, XNF1 est systématiquement moins performant, étant surpassé par les autres modélisations dans la majorité des cas. En comparant XNF2 aux modélisations CNF, CNF1 est plus rapide dans 47% des cas contre 37% en faveur de XNF2, et CNF2 l’emporte également dans 47% des cas contre 38% pour XNF2. Cette tendance est particulièrement marquée sur les instances de difficulté intermédiaire, où CNF1 et CNF2 dominent XNF2 respectivement dans 64% et 62% des cas. Entre les deux modèles CNF, CNF2 se montre légèrement supérieur, étant plus rapide que CNF1 dans 43% des cas contre 36% en faveur de CNF1. Cette domination de CNF2 est surtout visible sur les instances de difficulté intermédiaire et élevée.

Les modélisations CNF offrent globalement de meilleures performances que les représentations XNF. Parmi elles, CNF2 se démarque comme la plus performante, en particulier sur les instances de difficulté intermédiaire et élevée. À l’inverse, XNF1 affiche systématiquement les résultats les plus faibles. CNF1 et XNF2 obtiennent de meilleures performances que XNF1, mais restent globalement en retrait par rapport à CNF2 en terme de temps de résolution.

5 Conclusion

Dans cet article, nous avons présenté des travaux préliminaires visant à résoudre le problème du décodage par syndrome en utilisant des solveurs SAT, notamment CryptoMiniSat et CaDiCaL, en modélisant ce problème à l’aide de représentations XNF et CNF. Alors que les représentations XNF sont généralement reconnues comme plus efficaces que les représentations clausales pour les problèmes cryptographiques, nos résultats montrent que, dans le cadre du décodage par syndrome, les représentations CNF introduites offrent des performances comparables et compétitives à celles des représentations XNF.

Ces résultats ouvrent la voie à de nombreuses perspectives. Nos observations suggèrent qu’il serait pertinent d’explorer des stratégies d’encodage hybrides, combinant les avantages des représentations CNF et XNF, dans le but d’améliorer les performances globales, notamment sur les instances les plus complexes. De plus, il pourrait être judicieux d’expérimenter d’autres encodages pour les contraintes de cardinalité et pseudo-bouliennes, dans le but d’optimiser davantage les performances. Enfin, il serait intéressant d’étudier des modélisations des variantes du décodage par syndrome. En particulier, le décodage par syndrome avec poids faible⁴ s’intéresse à la recherche d’un vecteur d’erreur de poids minimal et on pourrait envisager de modéliser ce problème grâce à la satisfiabilité maximum (MaxSAT) [8], l’extension naturelle de SAT en problème d’optimisation.

4. <https://decodingchallenge.org/low-weight>

Remerciements

Ce travail est partiellement soutenu par le projet ANR-24-CE23-6126 (BforSAT) financé par l'agence nationale de recherche. Il a bénéficié d'un accès aux ressources HPC de la « Plateforme MatriCS » de l'Université de Picardie Jules Verne, qui est cofinancée par l'Union Européenne avec le Fonds Européen de Développement Régional (FEDER) et le Conseil Régional des Hauts-De-France entre autres.

Références

- [1] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks and their applications. In O. Kullmann, editor, *SAT 2009*, volume 5584 of *LNCS*, pages 167–180. Springer, 2009.
- [2] K. E. Batchner. Sorting networks and their applications. In *American Federation of Information Processing Societies - AFIPS 1968, Atlantic City, NJ, USA*, volume 32, pages 307–314. Thomson Book Company, Washington D.C., 1968.
- [3] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.
- [4] E. R. Berlekamp. *Algebraic coding theory*. McGraw-Hill series in systems science. McGraw-Hill, 1968.
- [5] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding : Turbo-codes. 1. In *ICC'93*, volume 2, pages 1064–1070. IEEE, 1993.
- [6] A. Biere. Pseudo-boolean and cardinality constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, chapter 22, pages 695–730. IOS Press, 2009.
- [7] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleys, and F. Pollitt. CaDiCaL 2.0. In A. Gurfinkel and V. Ganesh, editors, *CAV 2024*, volume 14681 of *LNCS*, pages 133–152. Springer, 2024.
- [8] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.
- [9] S. A. Cook. The complexity of theorem-proving procedures. In M. A. Harrison, R. B. Banerji, and J. D. Ullman, editors, *STOC 1971*, pages 151–158. ACM, 1971.
- [10] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman. Optimization with parity constraints : From binary codes to discrete integration. In A. E. Nicholson and P. Smyth, editors, *UAI 2013*. AUAI Press, 2013.
- [11] A. Esser and E. Bellini. Syndrome decoding estimator. In G. Hanaoka, J. Shikata, and Y. Watanabe, editors, *IACR 2022*, volume 13177 of *LNCS*, pages 112–141. Springer, 2022.
- [12] J. Feldman, M. J. Wainwright, and D. R. Karger. Using linear programming to decode binary linear codes. *IEEE Transactions on Information Theory*, 51(3):954–972, 2005.
- [13] R. Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.
- [14] S. Gocht and J. Nordström. Certifying parity reasoning efficiently using pseudo-boolean proofs. In *AAAI 2021*, pages 3768–3777. AAAI Press, 2021.
- [15] A. Gupta, M. K. Ganai, and C. Wang. Sat-based verification methods and applications in hardware verification. In M. Bernardo and A. Cimatti, editors, *SFM 2006, Advanced Lectures*, volume 3965 of *LNCS*, pages 108–143. Springer, 2006.
- [16] R. G. Jeroslow. On defining sets of vertices of the hypercube by linear inequalities. *Discrete Mathematics*, 11(2):119–124, 1975.
- [17] F. Lafitte, J. N. Jr., and D. V. Heule. Applications of SAT solvers in cryptanalysis : Finding weak keys and preimages. *J. Satisf. Boolean Model. Comput.*, 9(1):1–25, 2014.
- [18] N. Manthey, T. Philipp, and P. Steinke. A more compact translation of pseudo-boolean constraints into CNF such that generalized arc consistency is maintained. In C. Lutz and M. Thielscher, editors, *KI 2014*, volume 8736 of *LNCS*, pages 123–134. Springer, 2014.
- [19] R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, Jan. 1978.
- [20] C. A. Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, E. Persichetti, G. Zémor, and I. Bourges. Hamming quasi-cyclic (hqc). *NIST PQC Round*, 2(4):13, 2018.
- [21] S. Narisada, S. Uemura, H. Okada, H. Furue, Y. Aikawa, and K. Fukushima. Solving mceliece-1409 in one day - cryptanalysis with the improved BJMM algorithm. In N. Mouha and N. Nikiforakis, editors, *ISC 2024*, volume 15258 of *LNCS*, pages 3–23. Springer, 2024.
- [22] W. Nawrocki, Z. Liu, A. Fröhlich, M. J. H. Heule, and A. Biere. XOR local search for boolean brent equations. In C. Li and F. Manyà, editors, *SAT 2021*, volume 12831 of *LNCS*, pages 417–435. Springer, 2021.
- [23] J. Rintanen. Planning and SAT. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 765–789. IOS Press, 2021.
- [24] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.
- [25] J. P. M. Silva and K. A. Sakallah. GRASP - a new search algorithm for satisfiability. In R. A. Rutenbar and R. H. J. M. Otten, editors, *ICCAD 1996*, pages 220–227. IEEE Computer Society / ACM, 1996.
- [26] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT solvers to cryptographic problems. In O. Kullmann, editor, *SAT 2009*, volume 5584 of *LNCS*, pages 244–257. Springer, 2009.
- [27] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT solvers to cryptographic problems. In O. Kullmann, editor, *SAT 2009*, volume 5584 of *LNCS*, pages 244–257. Springer, 2009.
- [28] M. Trimoska, G. Dequen, and S. Ionica. Logical cryptanalysis with WDSat. In *Proceedings of SAT 2021*, Barcelona, Spain, July 2021.
- [29] M. Yannakakis. Expressing combinatorial optimization problems by linear programs. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 223–228, 1988.

**Session 4, mardi 1 juillet, 10h30-11h00 : PPC & Apprentissage
Automatique**

Apprentissage de précédences pour des problèmes de planification avec des réseaux de neurones en graphes

Hélène Verhaeghe^{1,2}, Quentin Cappart³, Gilles Pesant³, Claude-Guy Quimper⁴

¹ KULeuven, Louvain, Belgique

² UCLouvain, Louvain-la-Neuve, Belgique

³ Polytechnique Montréal, Montréal, Canada

⁴ Université Laval, Québec, Canada

helene.verhaeghe@uclouvain.be,quentin.cappart@polymtl.ca,gilles.pesant@polymtl.ca,
claude-guy.quimper@ift.ulaval.ca

Résumé

Le problème de gestion de projet à contraintes de ressources consiste à planifier un ensemble de tâches consommant des ressources durant un horizon temporel sujet à certaines capacités de ressources et relations de précédences entre des paires de tâches. Le problème est naturellement représenté par un graphe dirigé, généralement appelé graphe de précédences, où des paires de tâches soumises à une précédence sont liées. Dans ce papier, nous proposons d'utiliser les capacités d'un réseau de neurones en graphes pour extraire de la connaissance du graphe de précédence. Ce papier est un résumé de l'article "Learning Precedences for Scheduling Problems with Graph Neural Networks" [3] publié à CP2024.

Mots-clés

Planification, graphe de précédence, réseau de neurones en graphes.

Abstract

The resource constrained project scheduling problem (RCPSP) consists of scheduling a finite set of resource-consuming tasks within a temporal horizon subject to resource capacities and precedence relations between pairs of tasks. The problem is naturally represented as a directed graph, commonly referred to as the precedence graph, by linking pairs of tasks subject to a precedence. In this paper, we propose to leverage the ability of graph neural networks to extract knowledge from precedence graphs. This paper is a summary of the paper "Learning Precedences for Scheduling Problems with Graph Neural Networks" [3] published at CP2024.

Keywords

Scheduling, Precedence graph, Graph neural network.

1 Introduction

Les problèmes de planification existent dans de nombreux domaines, de l'assemblage d'avions à la planification de tâches de maintenance. La Programmation par Contraintes

(PPC) a déjà été utilisée de manière fructueuse pour résoudre différents types de problèmes de planification. Le succès de cette technique provient entre autres de la combinaison des contraintes globales et des heuristiques efficaces dédiées à ce type de problèmes. Les problèmes de planification sont souvent NP-difficiles quand ils sont sujets à des contraintes de précédence et de ressources. Cependant, celles-ci peuvent également aider à améliorer les inférences faites par les contraintes globales. Les précédences peuvent être naturellement représentées par un graphe dirigé en liant deux tâches sujettes à précédence. Les réseaux de neurones en graphes (RNG) sont conçus pour apprendre de données structurées en graphes, permettant, entre autre, d'apprendre à prédire si une arête est présente ou pas. Notre papier vise à répondre aux questions suivantes : un RNG peut-il nous aider à identifier de nouvelles précédences, et si oui, à quel point peuvent-elles être utiles ?

2 Méthodologie

Pour apprendre des précédences présentes dans les solutions, nous avons décidé d'utiliser un RNG à trois couches *GraphSAGE* [1], suivi d'un perceptron multicouche de deux couches (réseau entièrement connecté).

En pratique, pour obtenir une prédiction une fois le modèle entraîné, l'utilisateur crée le graphe d'entrée. Celui-ci a comme arêtes toutes les précédences de l'instance ainsi que la fermeture transitive de celles-ci. De plus, les noeuds sont annotés d'un vecteur contenant les caractéristiques de la tâche : la durée et la consommation normalisées de chaque ressource. De ce graphe, on peut automatiquement déduire les arêtes candidates (celle qui ne font pas partie de la fermeture transitive, ni de l'inverse de celle-ci). Le graphe est ensuite entré dans le RNG afin de calculer pour chaque noeud les projections des caractéristiques. Pour chaque arête candidate, les projections des deux extrémités sont ensuite entrées dans un perceptron multicouche, dont la tâche est de prédire la présence ou non de cette arête. Les prédictions sur les candidats sont ensuite classées par prédictions décroissantes et sélectionnées une à une si elles

ne créent pas de cycle et que leur prédiction a sa valeur au-dessus d'un certain seuil de sélection.

L'utilisateur peut ensuite faire ce qu'il désire des arêtes prédites. Dans notre cas, nous avons testé deux stratégies. Premièrement, nous créons une restriction du problème en ajoutant des contraintes pour toutes précédences prédites. Notre hypothèse est que cela peut diminuer la taille de l'arbre de recherche à explorer, mais pour chaque précédente mal prédite, il y a un risque d'enlever la partie de l'arbre contenant la solution optimale. Deuxièmement, à partir du graphe de précédence étendu des précédences prédites, nous pouvons créer un ordre topologique des tâches, qui peut être utilisé comme guide à une heuristique de recherche (telle que la combinaison sbps/vsids). Notre hypothèse dans cette situation est que si cet ordre est proche de celui de la solution optimale, alors celle-ci sera parmi les premières à être découverte. De plus, avec une heuristique, nous sommes sûrs de ne pas perdre la solution optimale.

3 Expérimentations

Nous avons utilisé le dépôt PSPLib [2] pour nos expériences. Les instances ont 30, 60, 90 ou 120 tâches. Les instances PSPLib sont des instances générées, avec différents paramètres (une série de 10 instances par ensemble de paramètres). Pour nous assurer de ne pas causer de surapprentissage, nous avons mis de côté certaines séries entières, ainsi que certaines instances des séries restantes. Pour chaque instance servant à l'apprentissage, nous avons calculé la meilleure solution après 1h de temps limite pour deux modèles PPC afin de comparer l'effet de la qualité de l'ensemble d'entraînement sur la méthode. Le premier utilise les heuristiques sbps/vsids, l'autre utilise une simple heuristique de placement de la tâche le plus tôt. Nous avons réalisé une validation croisée en K plis sur les arêtes à apprendre pour également éviter un surapprentissage.

3.1 Impact de la qualité de l'ensemble d'entraînement

Pour commencer, nous avons comparé un modèle entraîné sur les solutions générées avec sbps/vsids (solutions plus proches de l'optimal) et un modèle sur les solutions générées sans (moins proches de l'optimal). Les métriques d'entraînement étaient légèrement meilleures pour le second modèle. Cependant, lorsque nous les utilisons, soit comme contrainte additionnelle, soit comme guide heuristique, les prédictions du premier modèle se rendaient plus utiles. Cette expérience a également confirmé certaines hypothèses. Lorsque les prédictions sont utilisées comme contraintes additionnelles, elles créent une restriction, qui dans certains cas (si les prédictions sont correctes) permet d'améliorer les solutions pour un même temps de calcul. Cependant, dans une majorité des cas, la restriction diminue la qualité des résultats, dû à des prédictions erronées. Si les prédictions sont utilisées pour guider l'heuristique, nous n'avons en effet pas de perte de l'optimal, et, dans une majorité des instances, nous avons une amélioration de l'optimum au temps d'arrêt pour les plus faibles temps d'ar-

rêt, traduisant la découverte de meilleures solutions plus tôt dans la recherche.

3.2 Généralisation

La qualité de l'ensemble d'entraînement étant un facteur important sur la qualité des prédictions, idéalement il ne faudrait entraîner que sur des solutions optimales. Hors, plus il y a de tâches, plus il est difficile d'obtenir l'optimum en temps raisonnable. Nous avons donc mesuré la nécessité d'inclure des grosses instances dans l'ensemble d'entraînement. Pour ce faire, nous avons entraîné un modèle sur les instances de 60 tâches maximum et nous l'avons comparé aux performances du modèle entraîné sur les instances de 120 tâches ou moins. Les métriques des deux modèles étaient similaires.

3.3 Agrégation de solutions

Dans les problèmes de planification, il arrive souvent qu'il y ait plusieurs solutions optimales. Celles-ci diffèrent souvent légèrement l'une de l'autre (planning où des tâches moins contraintes et moins demandeuses en ressources ont un peu de liberté de placement). Pour encapsuler cette tendance, nous avons généré, pour chaque instance, 100 solutions avec le même optimum que le meilleur trouvé. L'agrégation de ces solutions correspond aux arêtes présentes dans au moins 70% de ces solutions. Notre but étant d'apprendre des précédences plus cruciales à l'optimalité. Les métriques d'apprentissage étaient similaires aux précédents modèles. Les métriques d'utilisation en étaient cependant améliorées, montrant une découverte de meilleures solutions plus tôt durant la recherche.

4 Conclusion

Ce papier présente une nouvelle approche basée sur les réseaux de neurones en graphes afin de prédire de nouvelles précédences pour le problème de gestion de projet à contraintes de ressources. Les précédences apprises peuvent soit être ajoutées comme contraintes additionnelles afin de réduire l'espace de recherche, soit comme heuristique afin de guider la recherche. Nos expériences sur les instances de PSPLIB nous confirment que dû à la nature difficile du problème, il est difficile d'obtenir un bon rappel au niveau des prédictions. Cependant, on observe quand même une accélération de la résolution. Une amélioration des solutions a été observée dans nos expériences mais reste difficile à atteindre. La qualité des prédictions dépend entre autres de la qualité de l'ensemble d'entraînement. Utiliser un agrégat de plusieurs solutions a permis un apprentissage de précédences plus cruciales aux solutions. Nos expériences montrent également une bonne généralisation, comme le montre la comparaison entre les modèles entraînés sur des instances de 60 tâches maximum et les modèles entraînés sur des instances de 120 tâches ou moins. Notre méthode est également agnostique du solveur et peut être combinée avec d'autres méta-heuristiques telles que la recherche à base de voisinage large.

Remerciements

Cette recherche a reçu du financement de la part de IVADO et des Fonds d'excellence en recherche Apogée Canada (PostDoc-2022-2378128196) et des fonds NSERC Discovery Grants 218028/2017, ainsi qu'un financement du programme Horizon 2020 de recherche et innovation de l'Union Européenne (No 101070149, projet Tuples).

Références

- [1] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.
- [2] Rainer Kolisch and Arno Sprecher. Psplib-a project scheduling problem library : Or software-orsep operations research software exchange program. *European journal of operational research*, 96(1) :205–216, 1997.
- [3] Hélène Verhaeghe, Quentin Cappart, Gilles Pesant, and Claude-Guy Quimper. Learning precedences for scheduling problems with graph neural networks. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, pages 30–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

Apprentissage de bornes duales valides en programmation par contrainte : Décomposition lagrangienne amplifiée avec apprentissage auto-supervisé

Swann Bessa^{1,2}, Darius Dabert^{1,2}, Max Bourgeat¹, Louis-Martin Rousseau¹, Quentin Cappart¹

¹ Polytechnique Montréal, Montréal, Canada

² École Polytechnique, Palaiseau, France

12 mai 2025

Résumé

Ce papier est un résumé de l'article "Learning Valid Dual Bounds in Constraint Programming : Boosted Lagrangian Decomposition with Self-Supervised Learning" publié à AAAI 2025. La décomposition lagrangienne relaxe les problèmes d'optimisation en sous-problèmes plus simples pour améliorer les algorithmes de séparation et d'évaluation en permettant le calcul d'une borne duale. Toutefois, en programmation par contraintes, l'optimisation des multiplicateurs de Lagrange est coûteuse en raison de la complexité de résolution des sous-problèmes. Nous proposons une approche d'apprentissage auto-supervisé utilisant des réseaux de neurones pour générer ces multiplicateurs et obtenir des bornes plus serrées. Cela réduit le nombre d'itérations nécessaires et accélère les solveurs. Notre méthode démontre une bonne généralisation sur des problèmes comme le sac à dos multidimensionnel.

1 Introduction

L'optimisation combinatoire est utilisée dans divers domaines comme l'aérospatial, la planification des transports, l'ordonnancement et l'économie. Un défi majeur est l'explosion combinatoire : le nombre de solutions possibles croît exponentiellement avec la taille du problème, rendant la résolution de problèmes à grande échelle difficile. La programmation par contraintes (CP) est un outil versatile qui permet de traiter des problèmes combinatoires complexes, y compris ceux avec des contraintes non linéaires, grâce à des techniques de recherche et de propagation [7]. Cependant, contrairement à la programmation en nombres entiers, la programmation par contraintes ne dispose pas d'un mécanisme efficace pour obtenir des bornes duales, limitant ainsi sa compétitivité sur de nombreux problèmes [2, 1, 8, 3]. Cet article propose une approche pour combler cette lacune en développant une méthode générale et efficace de bornage dual pour la programmation par contraintes.

2 Décomposition lagrangienne en CP

Un problème d'optimisation sous contraintes (COP) est défini par un ensemble de variables discrètes X , leurs do-

maines de valeurs $D(X)$, un ensemble de contraintes C limitant les affectations possibles, et une fonction objectif $f(X)$ à optimiser. Une solution faisable est une affectation des valeurs de $D(X)$ à X respectant toutes les contraintes. Une solution optimale est une solution faisable qui maximise (ou minimise) la fonction objectif. La formulation mathématique d'un COP inclut généralement m contraintes qui définissent l'espace des solutions admissibles.

$$\max_{X \in D(X)} \left\{ f(X) \mid \bigwedge_{i=1}^m C_i(X) \right\} \quad (1)$$

La notation $C_i(X)$ indique que la contrainte C_i s'applique aux variables X . La décomposition lagrangienne (LD) consiste à relaxer le problème afin d'obtenir une borne supérieure valide et, idéalement, serrée. Pour ce faire, les variables impliquées dans chaque contrainte sont dupliquées, à l'exception de celles de la première contrainte. On obtient alors la borne duale suivante :

$$\mathcal{B}(\mu) = \max \left\{ f(X_1) + \sum_{i=2}^m \mu_i \cdot (X_1 - X_i) \mid \bigwedge_{i=1}^m C_i(X_i) \right\} \quad (2)$$

où les μ_i sont les multiplicateurs de Lagrange. Le problème résultant est une relaxation du problème initial, fournissant une borne duale $\mathcal{B}(\mu)$, qui correspond à une borne supérieure dans un problème de maximisation. Chaque contrainte C_i ayant un ensemble de variables distinct, les sous-problèmes peuvent être résolus indépendamment, simplifiant ainsi la résolution. Ainsi, une borne duale valide est obtenue en résolvant m sous-problèmes plus simples, mais la qualité de cette borne dépend fortement des multiplicateurs de Lagrange utilisés. Trouver ces multiplicateurs optimaux est le principal défi de la décomposition lagrangienne, car l'équation associée n'est pas différentiable. La méthode la plus courante pour les ajuster est l'optimisation par sous-gradient, qui les met à jour de manière itérative à partir d'une valeur initiale arbitraire [9].

$$\mu_i^{t+1} = \mu_i^t + \alpha(X_1^t - X_i^t) \quad \forall i \in \{2, \dots, m\} \quad (3)$$

Hà et al. (2015) [4] ont proposé d'intégrer cette procédure dans la phase de recherche d'un solveur de programmation par contraintes (CP), en appliquant la décomposition lagrangienne (LD) et l'optimisation par sous-gradient à chaque nœud de l'arbre de recherche. Cependant, cette approche est très coûteuse en temps, car chaque itération du sous-gradient nécessite de résoudre chaque sous-problème, qui peuvent être NP-difficile. Nous proposons d'accélérer considérablement ce processus en utilisant de l'apprentissage automatique.

3 Notre Approche

La borne calculée dans l'Équation (2) possède deux propriétés importantes : (1) elle peut être paramétrée à l'aide des multiplicateurs de Lagrange μ et (2) elle est toujours valide, c'est-à-dire qu'elle ne sous-estimera jamais le profit réel. Ces propriétés ouvrent la voie à une approche basée sur l'apprentissage pour calculer cette borne. Inspirés par Parjadis et al. (2024) [6], nous proposons d'entraîner un modèle $\Omega_\theta : G(V, E) \rightarrow R^{|V|}$ capable de prédire directement tous les multiplicateurs μ pour un problème d'optimisation combinatoire (COP) donné sous forme de graphe G . Ce graphe est constitué d'un ensemble de nœuds V (un par multiplicateur) et d'arêtes E reliant les nœuds partageant la même variable ou la même contrainte. Le modèle Ω_θ est calibré par un ensemble de paramètres différentiables θ . L'objectif est d'éliminer les itérations du sous-gradient en apprenant directement les multiplicateurs qui produisent une borne serrée, ce qui permet d'accélérer l'exécution et d'améliorer le filtrage. La différentiabilité du modèle est essentielle, car la borne est calculée via une optimisation basée sur le gradient. L'objectif est de trouver des paramètres θ minimisant la borne. Grâce à la deuxième propriété, la borne obtenue a la garantie d'être valide, indépendamment de la précision du modèle. Ce travail introduit ainsi une méthode générique d'apprentissage de bornes duales valides pour tout type de COP discret, ce qui est un atout majeur étant donné les difficultés à obtenir des garanties en optimisation combinatoire avec l'apprentissage automatique [5]. La minimisation de la borne en fonction des multiplicateurs μ et son gradient sont ainsi formulés :

$$\min_{\theta} \mathcal{B}(\mu) \longrightarrow \nabla_{\theta} \mathcal{B}(\mu) \text{ avec } \mu = \Omega_{\theta}(G) \quad (4)$$

En appliquant la règle du chaînage on obtient alors :

$$\nabla_{\theta} \mathcal{B}(\mu) = \left\langle (X_1 - X_i) \cdot \frac{\partial \mu_i}{\partial \theta}, \dots \right\rangle \forall i \in \{2, \dots, m\} \quad (5)$$

Nous proposons d'utiliser une approche d'apprentissage auto-supervisé pour paramétrer le modèle Ω_θ . La procédure d'entraînement est formalisée comme suit. En entrée, elle reçoit un ensemble de données D composé de graphes $G(V, E)$ représentant des instances d'un problème d'optimisation combinatoire (COP). Ces instances peuvent provenir de données historiques ou être générées synthétiquement, et leurs caractéristiques dépendent du problème

traité. À chaque itération, une étape d'optimisation L_D est réalisée, et tous les sous-problèmes sont résolus via une procédure dédiée afin d'obtenir les valeurs optimales de X_i pour les multiplicateurs μ actuels. Ensuite, la borne et son gradient sont calculés, suivis d'une mise à jour par descente de gradient. Cela ajuste les valeurs des multiplicateurs pour l'itération suivante, modifiant ainsi les solutions optimales des variables X_i . Finalement, les paramètres θ du réseau de neurones entraîné Ω_θ sont retournés.

4 Expérimentation

La méthode est testée sur deux problèmes d'optimisation combinatoire difficiles : le problème du sac à dos multidimensionnel et le problème de planification des horaires. Nous testons différentes méthodes :

- **CP** : Une approche pure de programmation par contraintes sans décomposition lagrangienne ni apprentissage.
- **CP+SG** : Le même modèle CP, amélioré avec la décomposition lagrangienne et l'optimisation par sous-gradient, comme proposé par Hà et al. (2015) [4].
- **CP+Learning(all)** : Le modèle CP avec décomposition lagrangienne, utilisant les bornes apprises au lieu du sous-gradient, avec l'apprentissage appliqué à chaque nœud de l'arbre de recherche.
- **CP+Learning(all)+SG** : Similaire à l'approche précédente, mais les bornes apprises sont encore améliorées par l'optimisation sous-gradient. Le modèle entraîné Ω_θ est appelé à chaque nœud de l'arbre de recherche pour obtenir les bornes.
- **CP+Learning(root)+SG** : Le modèle Ω_θ est utilisé uniquement au nœud racine, la borne résultante servant de valeur initiale pour amorcer l'optimisation sous-gradient dans les autres nœuds.

Les résultats obtenus confirment que l'on obtient en moyenne des bornes de meilleures qualités et que notre modèle a une bonne capacité à s'adapter à des instances dont la distribution n'a pas été rencontrée durant la phase d'entraînement.

5 Conclusion

L'article propose une méthode innovante pour améliorer l'efficacité de la décomposition lagrangienne dans la programmation par contraintes. Il utilise un réseau de neurones pour apprendre automatiquement des multiplicateurs lagrangiens, permettant de produire des bornes duales serrées sans nécessiter de bornes étiquetées. Cette approche auto-supervisée permet soit de remplacer entièrement les itérations de sous-gradient, soit de les amorcer afin d'accélérer la convergence. Les expériences montrent que la méthode réduit le temps d'exécution tout en conservant des bornes de qualité. De plus, un ajustement fin permet une meilleure généralisation à des instances hors distribution. Ces résultats ouvrent des perspectives prometteuses pour appliquer cette technique à des problèmes combinatoires où la programmation par contraintes est performante.

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 :205–233, 2012.
- [2] Filippo Focacci, Andrea Lodi, and Michela Milano. Cost-based domain filtering. *Principles and Practice of Constraint Programming–CP’99 : 5th International Conference, CP’99, Alexandria, VA, USA, October 11-14, 1999. Proceedings 5*, 1999.
- [3] Monique Guignard and Siwhan Kim. Lagrangean decomposition : A model yielding stronger lagrangean bounds. *Mathematical programming*, 39(2) :215–228, 1987.
- [4] Minh Hoàng Hà, Claude-Guy Quimper, and Louis-Martin Rousseau. General bounding mechanism for constraint programs. In *Principles and Practice of Constraint Programming : 21st International Conference, CP 2015, Cork, Ireland, August 31–September 4, 2015, Proceedings 21*, pages 158–172. Springer, 2015.
- [5] James Kotary, Ferdinando Fioretto, Pascal van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning : A survey. In *30th International Joint Conference on Artificial Intelligence, IJCAI 2021*, pages 4475–4482. International Joint Conferences on Artificial Intelligence, 2021.
- [6] Augustin Parjadis, Quentin Cappart, Bistra Dilkina, Aaron Ferber, and Louis-Martin Rousseau. Learning Lagrangian Multipliers for the Travelling Salesman Problem. In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22 :1–22 :18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [7] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.
- [8] Meinolf Sellmann and Torsten Fahle. CP-based lagrangian relaxation for a multimedia application. In *Third International Workshop on the Integration of AI and OR Techniques (CPAIOR 2001)*, 2001.
- [9] Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*, volume 3. Springer Science & Business Media, 2012.

Session 5, mardi 1 juillet, 15h00-16h00 : MaxSAT

Approches exactes pour le problème de satisfiabilité diversifiée

Zhifei Zheng, Sami Cherif, Rui Sá Shibasaki, Chu-Min Li

Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

{zhifei.zheng;sami.cherif;rui.sa.shibasaki;chu-min.li}@u-picardie.fr

Résumé

La satisfiabilité diversifiée est un problème d'optimisation combinatoire qui cherche à identifier k modèles satisfaisant une formule CNF donnée tout en maximisant la somme des distances entre chaque paire de modèles. Cet article présente deux nouvelles méthodes exactes de résolution. La première approche s'appuie sur la programmation quadratique en nombres entiers, tandis que la seconde est basée sur une formulation pseudo-booléenne linéaire, pouvant également être adaptée à la satisfiabilité maximum (MaxSAT). Notre évaluation expérimentale approfondie démontre l'efficacité et la performance des deux méthodes proposées sur diverses instances. De plus, pour certaines familles d'instances, les approches proposées parviennent à obtenir des solutions optimales.

Mots-clés

Satisfiabilité diversifiée, Programmation en Nombres Entiers, Satisfiabilité Maximum, Modélisation

Abstract

Diverse Satisfiability is a combinatorial optimization problem that seeks to identify k models satisfying a given CNF formula while maximizing the sum of pairwise Hamming distances between them. This paper introduces two novel exact solution methods. The first approach relies on quadratic integer programming, while the second is based on a linear Pseudo-Boolean formulation, which can also be adapted to Maximum Satisfiability (MaxSAT). Our comprehensive experimental evaluation demonstrates the efficiency and effectiveness of both proposed methods across various benchmark instances. Furthermore, for some instance families, the proposed approaches manage to obtain optimal solutions.

Keywords

Diverse Satisfiability, Integer Programming, Maximum Satisfiability, Modeling

1 Introduction

La satisfiabilité propositionnelle (SAT) est un problème de décision consistant à déterminer si une formule en forme normale conjonctive (CNF) donnée peut être satisfaite par une affectation de variables [6], également appelée modèle de la formule. SAT est le premier à avoir été démontré NP-complet [8], et constitue un paradigme largement utilisé pour la modélisation de nombreux problèmes du

monde réel, notamment en vérification matérielle et logicielle [1, 7], en planification et ordonnancement [24, 25], ou encore en bio-informatique [16, 17]. La version classique du problème SAT vise à trouver un unique modèle pour une formule CNF donnée, mais de nombreuses applications réelles exigent l'identification de plusieurs modèles présentant des propriétés structurelles diverses. Dans ce cas, le fait de s'arrêter au premier modèle ne permet pas de mener à bien l'analyse souhaitée ou risque l'omission d'informations cruciales. Un exemple représentatif qu'on peut citer est celui de la vérification bornée de modèles (Bounded Model Checking, BMC) [1], où chaque modèle de la formule correspond à un chemin de vérification possible. Toutefois, se limiter à une unique solution peut accroître le risque d'omettre des erreurs spécifiques, dans la mesure où celle-ci ne permet pas de capturer l'ensemble des comportements potentiels du système vérifié. Cette limitation met en évidence la nécessité de générer plusieurs solutions présentant une forte diversité structurelle, en vue d'accroître la robustesse des processus de débogage et de vérification.

Dans cette optique, Nadel a introduit le problème de la satisfiabilité diversifiée (Diverse Satisfiability) dans [20]. Étant donné une formule CNF, ce problème consiste à identifier k modèles de la formule maximisant la diversité structurelle. Celle-ci est quantifiée à l'aide de la somme des distances de Hamming entre toutes les paires de modèles sélectionnés, assurant ainsi une dispersion maximale dans l'espace des solutions. Dans ce même travail, l'auteur propose une méthode incomplète s'appuyant sur une heuristique de décision dédiée. Un problème connexe introduit dans la littérature est celui de la distance de Hamming maximale pour les problèmes de satisfaction de contraintes (CSP), initialement formulé par Crescenzi et Rossi dans [9]. Néanmoins, ce problème, bien qu'applicable à des instances à domaines finis, se limite à la recherche de deux modèles distincts. Dès lors, dans le cas binaire, il devient essentiellement équivalent à une instance de la satisfiabilité diversifiée avec $k = 2$.

Étant donné que le problème SAT est déjà NP-complet, le problème de la satisfiabilité diversifiée, qui requiert l'identification de k modèles satisfaisants distincts, représente un défi computationnel encore plus complexe. En particulier, pour un paramètre donné p , Misra et al. [19] démontrent que, même dans le cas restreint où $k = 2$, déterminer deux modèles qui diffèrent exactement (ou au moins) sur p variables demeure un problème NP-difficile. Dans [2], An-

gelsmark et Thapper établissent une borne supérieure de complexité en $O(1.7338^n)$ pour la résolution du problème de distance de hamming maximale sur les formules 2-SAT (c'est-à-dire la satisfiabilité diversifiée avec $k = 2$, où l'instance ne contient que des clauses de longueur 2). Plus généralement, ils montrent que, pour une formule l -SAT où l désigne la longueur maximale des clauses, une borne supérieure en $O((2a)^n)$ peut être obtenue, sous l'hypothèse que l'instance sous-jacente de l -SAT peut être résolue en $O(a^n)$ pour une certaine constante a .

Dans cet article, nous nous focalisons sur l'introduction d'approches exactes pour le problème de satisfiabilité diversifiée, lesquelles, à notre connaissance, n'ont pas encore été explorées dans la littérature. Nous proposons deux approches génériques, l'une reposant sur la programmation entière quadratique (QIP) et l'autre sur la programmation pseudo-boléenne linéaire, cette dernière pouvant être aisément transformée en une instance de satisfiabilité maximum (MaxSAT), l'extension naturelle de SAT en problème d'optimisation. Notre approche QIP permet de formuler la fonction objectif de manière naturelle sous forme quadratique, tandis que les approches linéaires reformulent l'objectif comme une combinaison linéaire d'une série de variables booléennes, s'appuyant sur deux fonctions de pondération : la pondération directe (DW) et la pondération incrémentale (IW). L'efficacité des approches proposées est ensuite évaluée au travers d'expérimentations approfondies.

La suite de cet article est structurée comme suit. La section 2 présente les définitions formelles, les notations ainsi que les fondements théoriques du problème de satisfiabilité diversifiée, en plus d'introduire les paradigmes standards de la programmation entière quadratique, la programmation pseudo-boléenne linéaire, et la satisfiabilité maximum. La section 3 expose nos approches exactes, incluant la formulation quadratique ainsi que les formulations pseudo-boléennes linéaires et leur transformation en modèles MaxSAT. La section 4 fournit une analyse détaillée des résultats expérimentaux obtenus. Enfin, la section 5 récapitule nos contributions et discute les perspectives de nos travaux.

2 Préliminaires

2.1 Satisfiabilité diversifiée

Soit X un ensemble de variables booléennes prenant des valeurs dans $\{Vrai, Faux\}$ (ou $\{0, 1\}$). Un littéral l est soit une variable $x \in X$, soit sa négation $\bar{x}$. Le littéral positif $l = x$ est dit satisfait lorsque x est affectée à *Vrai*, tandis que le littéral négatif $l = \bar{x}$ est satisfait lorsque x est affectée à *Faux*. Une clause C est une disjonction ($\vee$) de littéraux, et peut également être représentée comme un ensemble de littéraux. Plus précisément, on peut écrire un clause sous la forme $C = C^+ \cup C^-$, où C^+ est l'ensemble des littéraux positifs et C^- l'ensemble des littéraux négatifs. De plus, une clause est satisfaite dès lors que l'un de ses littéraux est satisfait. Une formule en forme normale conjonctive (CNF) est une conjonction ($\wedge$) de clauses, et elle est satisfaite si toutes ses clauses sont sa-

tisfaites. Une formule CNF ϕ composée de m clauses peut être représentée comme un ensemble de clauses, c'est-à-dire $\phi = \{C_1, \dots, C_m\}$. Étant donnée une formule CNF ϕ définie sur les variables de X , le problème de satisfiabilité propositionnelle (SAT) consiste à déterminer s'il existe une affectation $\alpha : X \rightarrow \{Vrai, Faux\}$ qui satisfait ϕ . Lorsqu'une telle affectation existe, on l'appelle modèle de ϕ et on dit que la formule est satisfiable.

Nous introduisons ci-après de manière formelle la notion de distance de Hamming entre deux affectations, qui correspond au nombre de valeurs différentes attribuées aux variables booléennes dans une paire d'affectations. Nous définissons ensuite une notion essentielle à la formulation du problème de satisfiabilité diversifiée, à savoir la mesure de *diversité* pour des ensembles d'affectations.

Définition 1 (Distance de Hamming entre affectations). *Étant donné un ensemble de variables booléennes X et deux affectations α_1, α_2 des variables de X , la distance de Hamming entre α_1 et α_2 , notée $Dis(\alpha_1, \alpha_2)$, est définie comme suit :*

$$Dis(\alpha_1, \alpha_2) = |\{x \in X \mid \alpha_1(x) \neq \alpha_2(x)\}|$$

$$= \sum_{x \in X} |\alpha_1(x) - \alpha_2(x)|$$

Définition 2 (Diversité). *Étant donné un ensemble de variables booléennes X et un ensemble $A = \{\alpha_1, \dots, \alpha_k\}$ de k affectations des variables dans X , la diversité de A , notée $Div(A)$, est définie comme suit :*

$$Div(A) = \sum_{i=1}^{k-1} \sum_{j=i+1}^k Dis(\alpha_i, \alpha_j) \quad (1)$$

Étant donnée une formule CNF ϕ et un entier $k \geq 2$, le problème de la satisfiabilité diversifiée (Diverse satisfiability, ou Diverse SAT) consiste à trouver un ensemble de k modèles de ϕ avec une diversité maximale. Plus formellement, nous cherchons :

$$A = \operatorname{argmax}_{A \in \{0,1\}^{k \times |X|}} Div(A)$$

Dans [20], Nadel évoque la notion de diversité pour une variable, c'est-à-dire le nombre de ses assignations à *Vrai* multiplié par le nombre de ses assignations à *Faux*, telle que définie formellement ci-dessous. Cette notion lui a servi de critère pour évaluer la contribution d'une seule variable à la diversité, dans l'heuristique proposée par l'auteur. Nous formalisons la correspondance entre la diversité classique et celles des variables dans Proposition 1.

Définition 3 (Diversité des Variables). *Étant donné un ensemble de variables booléennes X et un ensemble $A = \{\alpha_1, \dots, \alpha_k\}$ de k affectations des variables de X , la diversité de la variable $x \in X$ dans A , notée $Div(x, A)$, est définie comme suit :*

$$Div(x, A) = T(x, A) \cdot F(x, A)$$

où :

- $T(x, A) = |\{\alpha \in A \mid \alpha(x) = 1\}|$,
- $F(x, A) = |\{\alpha \in A \mid \alpha(x) = 0\}|$.

Proposition 1. *Étant donné un ensemble de variables booléennes X et un ensemble $A = \{\alpha_1, \dots, \alpha_k\}$ de k affectations des variables dans X , on a :*

$$Div(A) = \sum_{x \in X} Div(x, A) \quad (2)$$

Preuve.

$$\begin{aligned} Div(A) &= \sum_{i=1}^{k-1} \sum_{j=i+1}^k Dis(\alpha_i, \alpha_j) \\ &= \sum_{i=1}^{k-1} \sum_{j=i+1}^k |\{x \in X \mid \alpha_i(x) \neq \alpha_j(x)\}| \\ &= \sum_{i=1}^{k-1} \sum_{j=i+1}^k \sum_{\substack{x \in X \\ \alpha_i(x) \neq \alpha_j(x)}} 1 \\ &= \sum_{x \in X} \sum_{i=1}^{k-1} \sum_{\substack{j=i+1 \\ \alpha_i(x) \neq \alpha_j(x)}}^k 1 \\ &= \sum_{x \in X} T(x, A) \cdot F(x, A) = \sum_{x \in X} Div(x, A) \end{aligned}$$

□

Dans le reste de cet article, nous calculons la diversité des modèles à l'aide de l'équation 2 puisqu'elle permet de représenter plus naturellement le problème dans nos formulations.

2.2 Programmation quadratique en nombres entiers

La programmation quadratique en nombres entiers (Quadratic Integer Programming, QIP) est un paradigme de programmation mathématique avec des variables entières, une fonction objectif quadratique et des contraintes linéaires. Comme défini dans [22], la forme classique d'un modèle QIP est la suivante :

$$\begin{aligned} (QIP) \quad & \max_x x^\top H x + c^\top x \\ & \text{s.t. } x \in \mathcal{P} \\ & x \in \mathbb{Z}^n \end{aligned}$$

où $H \in \mathbb{Q}^{n \times n}$ est une matrice symétrique, $c \in \mathbb{Q}^n$ et $\mathcal{P}$ est un polyèdre, représenté par $\mathcal{P} = \{x : Ax \leq b\}$ où $A \in \mathbb{Q}^{m \times n}$ et $b \in \mathbb{Q}^m$.

La programmation pseudo-booléenne linéaire (Linear Pseudo-Boolean Programming, LPB) est définie dans le domaine des variables booléennes. Selon [10], elle peut être formulée sous le format suivant :

$$\begin{aligned} (LPB) \quad & \max f(x_1, \dots, x_n) \\ & \text{s.t. } f_j(x_1, \dots, x_n) \geq 0, \quad j \in \{1, \dots, m\} \\ & x \in \{0, 1\}^n \end{aligned}$$

où $f(x_1, \dots, x_n) = \sum_{i=1}^n w_i \cdot x_i$, $w_i \in \mathbb{Z}$.

2.3 Satisfiabilité maximum

La satisfiabilité maximum (MaxSAT) est l'extension naturelle de SAT en problème d'optimisation. Dans cet article, nous utilisons sa version la plus générique, à savoir MaxSAT partiel pondéré [4, 15]. Ce problème prend en entrée une formule bipartite pondérée $\phi = H \cup S$ contenant des variables booléennes dans X . H est l'ensemble des clauses dures qui doivent être satisfaites, comme dans SAT. S est l'ensemble des clauses souples, composé de clauses pondérées (C, W_C) , où W_C est un poids entier positif associé à la clause C . L'objectif de MaxSAT partiel pondéré est d'obtenir une affectation α qui maximise (resp. minimise) la somme des poids des clauses souples satisfaites (resp. falsifiées) tout en satisfaisant toutes les clauses dures. Formellement, soit $cost_\alpha(\phi)$ la somme des poids des clauses souples de ϕ falsifiées par l'affectation α , le problème MaxSAT partiel pondéré consiste ainsi à déterminer $optimum(\phi) = \min_\alpha cost_\alpha(\phi)$.

Dans le reste de l'article, nous utilisons MaxSAT pour faire référence à sa variante partielle pondérée. De plus, la littérature introduit des contraintes pseudo-boléennes (PB) qui peuvent être efficacement encodées sous forme clause [23]. Ces contraintes prennent la forme suivante :

$$\sum_j a_j \cdot l_j \triangleright b$$

où $a_j \in \mathbb{N}$, $b \in \mathbb{N}$, l_j est un littéral et $\triangleright \in \{=, \geq, \leq\}$. Un cas particulier de contrainte PB survient lorsque tous les coefficients des littéraux sont égaux à un et peuvent donc être omis [23]. Ces contraintes sont couramment appelées contraintes de cardinalité et prennent donc la forme la suivante :

$$\sum_j l_j \bowtie b$$

où l_j est un littéral, $b \in \mathbb{N}$ et $\bowtie \in \{=, \leq, \geq\}$.

3 Formulations pour la satisfiabilité diversifiée

Dans cette section, nous présentons trois formulations distinctes du problème de satisfiabilité diversifiée, à savoir une formulation basée sur un modèle de programmation quadratique en nombres entiers (QP) et deux formulations linéaires pseudo-boléennes (DW et IW). Nous commençons par la première formulation QP, présentée en 3.1, qui transforme naturellement le problème d'optimisation initial en une fonction objectif quadratique. En 3.2, nous exposons l'idée générale permettant de faire le lien entre une formulation avec variables entières et objectif quadratique, et une formulation avec variables booléennes et objectif linéaire, conduisant ainsi aux deux modèles linéaires proposés en 3.3. Enfin, la section 3.4 présente la transformation des formulations linéaires pseudo-boléennes vers des formulations MaxSAT.

3.1 Formulation quadratique

Dans cette sous-section, nous présentons une formulation en programmation quadratique en nombres entiers (QIP)

$$\max \sum_{j \in N} O_j \cdot (k - O_j) \quad (\text{QP-0})$$

s.c

$$\sum_{l_j \in C_h^-} (1 - V_{i,j}) + \sum_{l_j \in C_h^+} V_{i,j} \geq 1 \quad \forall (i, h) \in K \times M \quad (\text{QP-1})$$

$$O_j = \sum_{i=\{1, \dots, K\}} V_{i,j} \quad \forall j \in N \quad (\text{QP-2})$$

$$O_j \in \mathbb{N}^+, V_{i,j} \in \{0, 1\} \quad \forall (i, j) \in K \times N \quad (\text{QP-3})$$

FIGURE 1 – Modèle quadratique pour la satisfiabilité diversifiée

pour le problème de satisfiabilité diversifiée. Cette formulation s'appuie naturellement sur la Proposition 1, qui exprime la diversité comme la somme des diversités individuelles des variables. Étant donnée une formule CNF ϕ contenant n variables et m clauses, notre objectif est d'obtenir k modèles maximisant la diversité. Pour cela, nous définissons un modèle QIP à l'aide des deux séries de variables suivantes, où $K = \{1, \dots, k\}$, $N = \{1, \dots, n\}$ et $M = \{1, \dots, m\}$:

- **Variables entières** $O_j \quad \forall j \in N$, qui représentent le nombre d'affectations à *Vrai* de la $j^{\text{ème}}$ variable dans l'ensemble A des k affectations, c'est à dire $T(x_j, A)$.
- **Variables binaire** $V_{i,j} \quad \forall (i, j) \in K \times N$, qui représentent la valeur booléenne de la $j^{\text{ème}}$ variable dans la $i^{\text{ème}}$ affectation $\alpha_i \in A$, c'est à dire $\alpha_i(x_j)$.

Nous obtenons la formulation quadratique (QP) illustrée dans la Figure 1. L'équation QP-0 est une reformulation de Proposition 1, indiquant que l'objectif de maximisation, à savoir la diversité, est égal à la somme des diversités individuelles des variables. L'équation QP-1 garantit que l'ensemble A des k affectations construites constitue bien des modèles de la formule CNF ϕ donnée en entrée. L'équation QP-2 établit le lien sémantique entre les variables booléennes $V_{i,j}$ et les variables entières O_j . Cette formulation requiert $O(k \cdot n)$ variables et $O(k \cdot m + n)$ contraintes.

3.2 De la formulation quadratique aux formulations linéaires

Dans la sous-section précédente, la formulation proposée utilise des variables entières ainsi qu'une fonction objectif quadratique, conformément à la relation établie dans Proposition 1. Toutefois, la présence d'un terme quadratique dans la fonction objectif peut constituer un obstacle important à la résolution efficace du problème. Par conséquent, nous proposons une approche alternative consistant à transformer la formulation QIP en une formulation linéaire pseudo-booléenne. Autrement dit, il s'agit de transformer la fonction objectif quadratique initiale en une combinaison linéaire de variables booléennes.

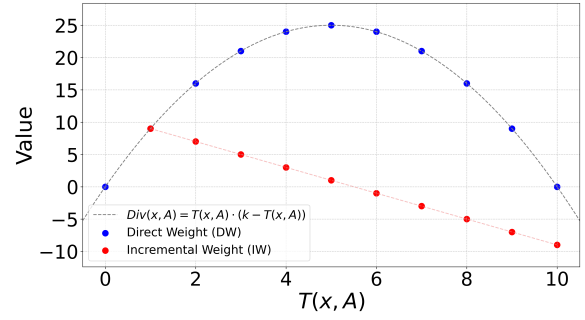


FIGURE 2 – Les fonctions de poids DW et IW pour un scénario avec $k = 10$

Pour chaque variable booléenne d'origine $x_j \in X$ dans la formule CNF ϕ donnée, nous introduisons d'abord une série de variables booléennes auxiliaires destinées à représenter le nombre d'affectations à *Vrai* de x_j à travers les k modèles, soit $T(x_j, A)$. Il convient de souligner que la relation de correspondance entre ces variables auxiliaires et l'entier $T(x_j, A)$ doit être cohérente avec les fonctions de pondération. Nous développons ensuite deux fonctions de pondération distinctes, appelées respectivement *pondération directe* (DW) et *pondération incrémentale* (IW), permettant de reformuler le terme quadratique original $Div(x_j, A) = T(x_j, A) \cdot (k - T(x_j, A))$.

Une illustration de ces deux fonctions de pondération dans un scénario comportant $k = 10$ modèles est présentée dans la Figure 2. La première fonction, DW , associe directement à chaque valeur possible de $T(x_j, A)$ sa contribution correspondante à la diversité. Les points bleus répartis sur la courbe parabolique indiquent que le maximum de diversité est atteint lorsque x_j est assignée à *Vrai* dans exactement la moitié des modèles. La seconde fonction, IW , adopte une approche incrémentale en exprimant $Div(x_j, A)$ comme la somme de contributions marginales. Les points rouges de la Figure 2 illustrent la variation de la valeur de diversité lorsqu'on incrémente $T(x_j, A)$ d'une unité. Il est à noter que ces valeurs incrémentales deviennent négatives au-delà du point médian, traduisant les rendements décroissants associés à une répartition déséquilibrée des affectations à *Vrai*. Formellement, soit $r = T(x_j, A)$, nous définissons la fonction de *pondération directe* (DW) comme suit :

$$\mathcal{D}_r = r \cdot (k - r) \quad \forall r \in \{0, \dots, k\}$$

et la fonction de *pondération incrémentale* (IW) par :

$$\mathcal{I}_r = \mathcal{D}_r - \mathcal{D}_{r-1} = -2r + k + 1 \quad \forall r \in \{1, \dots, k\}$$

On peut remarquer que pour tout $r \in \{1, \dots, k\}$, on a $\mathcal{D}_r = \sum_{r'=1}^r \mathcal{I}_{r'}$. Ces deux fonctions de pondération distinctes offrent donc des approches différentes mais mathématiquement équivalentes pour reformuler la fonction objectif d'origine, comme cela sera détaillé dans la suite.

$$\max \sum_{j=1}^n \sum_{r=0}^k \mathcal{D}_r \cdot U_{j,r} \quad (\text{DW-0})$$

s.c

$$\sum_{l_j \in C_h^-} (1 - V_{i,j}) + \sum_{l_j \in C_h^+} V_{i,j} \geq 1 \quad \forall (i, h) \in K \times M \quad (\text{DW-1})$$

$$\sum_{r \in (\{0\} \cup K)} U_{j,r} = 1 \quad \forall j \in N \quad (\text{DW-2})$$

$$\sum_{r \in (\{0\} \cup K)} r U_{j,r} = \sum_{i \in K} V_{i,j} \quad \forall j \in N \quad (\text{DW-3})$$

$$V_{i,j} \in \{0, 1\}, U_{j,r} \in \{0, 1\} \quad \forall (i, j) \in K \times N \\ \forall (r, j) \in (\{0\} \cup K) \times N \quad (\text{DW-4})$$

FIGURE 3 – Modèle linéaire pour la satisfiabilité diversifiée avec le pondération directe DW .

3.3 Formulations linéaires

3.3.1 Formulation par pondération directe

Nous présentons d'abord la formulation pour le problème de satisfiabilité diversifiée avec la fonction de pondération DW , qui repose sur les variables suivantes, où $K = \{1, \dots, k\}$ et $N = \{1, \dots, n\}$:

- **Variables binaires** $U_{j,r}$ $\forall (r, j) \in (\{0\} \cup K) \times N$, qui représentent le fait que le nombre d'affectations à *Vrai* de la $j^{\text{ème}}$ variable dans l'ensemble des k modèles soit égal à r , c'est à dire $T(x_j, A) = r$.
- **Variables binaire** $V_{i,j}$ $\forall (i, j) \in K \times N$, qui représentent la valeur booléenne de la $j^{\text{ème}}$ variable dans la $i^{\text{ème}}$ affectation $\alpha_i \in A$, c'est à dire $\alpha_i(x_j)$.

Ainsi, la formulation avec la fonction de pondération DW , $\mathcal{D}_r = r \cdot (k - r)$, peut être obtenue comme indiqué dans la Figure 3, où $M = \{1, \dots, m\}$. L'équation **DW-0** formule la fonction objectif, où chaque terme $\mathcal{D}_r \cdot U_{j,r}$ indique qu'il y a une contribution à la diversité $\mathcal{D}_r = r \cdot (k - r)$ lorsque la variable x_j est assignée à *Vrai* exactement dans r des k modèles. L'équation **DW-1** garantit que chaque modèle α_i satisfait la formule CNF originale ϕ , c'est-à-dire que pour toute clause C_h de ϕ et chaque modèle α_i , au moins un littéral de la clause doit être satisfait selon les assignations de variables dans α_i . L'équation **DW-2** impose qu'il y ait un seul nombre d'assignations à *Vrai* de la variable j parmi les k modèles. L'équation **DW-3** établit la relation de correspondance entre les variables $U_{j,r}$ et les assignations réelles des modèles $V_{i,j}$. Il y a $O(k \cdot n)$ variables et $O(k \cdot m + n)$ contraintes dans la formulation DW .

3.3.2 Formulation par pondération incrémentale

Dans cette sous-section, nous développons une autre formulation pour le problème de satisfiabilité diversifiée en utilisant la fonction IW . Les variables utilisées dans cette formulation sont les suivantes :

- **Variables binaires** $U_{j,r}$ $\forall (r, j) \in (\{0\} \cup K) \times N$, qui représentent le fait que le nombre d'affectations à *Vrai* de la $j^{\text{ème}}$ variable dans l'ensemble des k af-

$$\max \sum_{j=1}^n \sum_{r=0}^k \mathcal{I}_r \cdot U_{j,r} \quad (\text{IW-0})$$

s.c

$$\sum_{l_j \in C_h^-} (1 - V_{i,j}) + \sum_{l_j \in C_h^+} V_{i,j} \geq 1 \quad \forall (i, h) \in K \times M \quad (\text{IW-1})$$

$$U_{j,r} \leq U_{j,r-1} \quad \forall (r, j) \in (K \setminus \{1\}) \times N \quad (\text{IW-2})$$

$$\sum_{r \in K} U_{j,r} = \sum_{i \in K} V_{i,j} \quad \forall j \in N \quad (\text{IW-3})$$

$$V_{i,j} \in \{0, 1\}, U_{j,r} \in \{0, 1\} \quad \forall (i, j) \in K \times N \\ \forall (r, j) \in K \times N \quad (\text{IW-4})$$

FIGURE 4 – Modèle linéaire pour la satisfiabilité diversifiée avec le pondération incrémentale IW .

fectations A soit supérieure ou égale à r , c'est à dire $T(x_j, A) \geq r$.

- **Variables binaire** $V_{i,j}$ $\forall (i, j) \in K \times N$, qui représentent la valeur booléenne de la $j^{\text{ème}}$ variable dans la $i^{\text{ème}}$ affectation $\alpha_i \in A$, c'est à dire $\alpha_i(x_j)$.

La formulation linéaire avec IW , définie par $\mathcal{I}_r = -2r + k + 1$, est obtenue comme illustré dans la Figure 4. La structure de IW reste similaire à celle de DW . L'équation **IW-0** exprime la contribution incrémentale de l'affectation *Vrai* de la variable x_j à la diversité globale. L'équation **IW-1** impose la satisfiabilité de l'ensemble des k affectations obtenues. L'équation **IW-2** assure l'ordre dans l'utilisation des variables $U_{j,r}$ ce qui, en conjonction avec **IW-3**, permet de capturer l'information sémantique de $U_{j,r}$. Les complexités en nombre de variables et de contraintes dans la formulation IW sont respectivement de $O(n \cdot k)$ et $O(k \cdot (m + n))$.

3.4 Formulations MaxSAT

Lorsqu'on se focalise sur l'exigence intrinsèque du problème de satisfiabilité diversifiée, on ne peut pas ignorer la tâche centrale consistant à obtenir des modèles en logique propositionnelle, une tâche pour laquelle MaxSAT semble intuitivement être mieux adapté. Dans cette sous-section, nous présentons la transformation des formulations linéaires précédentes vers MaxSAT. Nous reformulons les modèles sous forme clausale, accompagnée de contraintes pseudo-bouliennes (PB) ou de cardinalité, comme illustré dans la Figure 5. De plus, nous signalons que l'équation **IW-0*** peut contenir des poids négatifs, ce qui n'est pas conforme au standard original de MaxSAT. Elle est donc transformée à l'aide de l'équation suivante :

$$\begin{cases} (U_{j,i}, \mathcal{I}_r) & \mathcal{I}_r \geq 0, \\ (\bar{U}_{j,i}, -\mathcal{I}_r) & \mathcal{I}_r < 0, \end{cases} \quad \forall (r, j) \in K \times N \quad (\text{IW-0})$$

Un exemple illustratif est présenté dans l'exemple suivant. Cette technique de normalisation permet de représenter correctement la fonction de poids incrémentale dans le cadre standard de MaxSAT, qui exige des poids non négatifs pour toutes les clauses souples.

Exemple 1. Si on prend $k = 4$ and $n = 3$, selon la définition $\mathcal{I}_r = -2r + k + 1$, on a :

$$\mathcal{I}_1 = 3, \mathcal{I}_2 = 1, \mathcal{I}_3 = -1, \mathcal{I}_4 = -3$$

Pour chaque variable x_j où $j \in \{1, 2, 3\}$, avant normalisation, les clause souples présentent des poids négatifs selon *IW-0** :

$$(U_{j,1}, 3), (U_{j,2}, 1), (U_{j,3}, -1), (U_{j,4}, -3)$$

La contribution des x_j peut être formulée comme suit :

$$obj_{original} = 3U_{j,1} + U_{j,2} - U_{j,3} - 3U_{j,4}$$

Après normalisation de *IW-0**, on obtient :

$$(U_{j,1}, 3), (U_{j,2}, 1), (\bar{U}_{j,3}, 1), (\bar{U}_{j,4}, 3)$$

avec la contribution suivante :

$$obj_{normalized} = 3U_{j,1} + U_{j,2} + \bar{U}_{j,3} + 3\bar{U}_{j,4}$$

Puisque $\bar{U}_{j,r} = 1 - U_{j,r}$, on peut réécrire les objectifs comme suit :

$$\begin{aligned} obj_{normalized} &= 3U_{j,1} + U_{j,2} + (1 - U_{j,3}) + 3(1 - U_{j,4}) \\ &= 3U_{j,1} + U_{j,2} - U_{j,3} - 3U_{j,4} + 4 \\ &= obj_{original} + 4 \end{aligned}$$

L'ajout d'une constante à toutes les valeurs de la fonction objectif ne modifie pas la solution optimale d'une instance MaxSAT donc la formulation normalisée permet bien d'identifier les mêmes ensembles optimaux de modèles diversifiés.

4 Résultats expérimentaux

Dans cette section, nous présentons les résultats expérimentaux des approches proposées. Le protocole expérimental, incluant l'environnement de test, les instances traitées, ainsi que les solveurs utilisés, est détaillé en 4.1. La section 4.2 présente les performances globales, tandis que la section 4.3 compare les formulations basées sur les deux fonctions de pondération distinctes.

4.1 Protocole expérimental

Tous les tests ont été menés sur la plateforme MatriCS¹, équipée d'un système CentOS 8.6 et d'un processeur Intel Xeon E5-2680 v4 fonctionnant à une fréquence de 2,40 GHz, avec une capacité Turbo Boost allant jusqu'à 3,30 GHz. Le temps limite fixé pour tous les solveurs est de 7200 secondes. Les instances utilisées dans nos expériences proviennent de deux sources : 20 instances issues de la vérification de matériel, telles que proposées et utilisées dans [20], ainsi que 88 instances provenant de SATLIB², pour un total de 108 instances réparties en 8 familles distinctes. La taille de ces instances varie de 61 variables et 300 clauses

1. <https://www.matrics.u-picardie.fr>

2. <https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html>

Soft Clauses

$$(U_{j,r}, r \cdot (k - r)) \quad \forall (r, j) \in (\{0\} \cup K) \times N \quad (\text{DW-0})$$

Hard Clauses

$$\left(\bigvee_{l_j \in C_h^-} \bar{V}_{i,j} \right) \vee \left(\bigvee_{l_j \in C_h^+} V_{i,j} \right) \quad \forall (i, h) \in K \times M \quad (\text{DW-1})$$

$$\sum_{r \in (\{0\} \cup K)} U_{j,r} = 1 \quad \forall j \in N \quad (\text{DW-2})$$

$$\sum_{r \in (\{0\} \cup K)} r U_{j,r} = \sum_{i \in K} V_{i,j} \quad \forall j \in N \quad (\text{DW-3})$$

(a) MaxSAT-DW

Soft Clauses

$$(U_{j,r}, \mathcal{I}_r) \quad \forall (r, j) \in K \times N \quad (\text{IW-0*})$$

Hard Clauses

$$\left(\bigvee_{l_j \in C_h^-} \bar{V}_{i,j} \right) \vee \left(\bigvee_{l_j \in C_h^+} V_{i,j} \right) \quad \forall (i, h) \in K \times M \quad (\text{IW-1})$$

$$U_{j,r} \rightarrow U_{j,r-1} \quad \forall (r, j) \in (K \setminus \{1\}) \times N \quad (\text{IW-2})$$

$$\sum_{r \in K} U_{j,r} = \sum_{i \in K} V_{i,j} \quad \forall j \in N \quad (\text{IW-3})$$

(b) MaxSAT-IW

FIGURE 5 – Formulations MaxSAT avec DW (a) et IW (b).

à 83908 variables et 276106 clauses. Toutes les instances garantissent l'existence d'au moins 10 modèles distincts.

Les solveurs de l'état de l'art suivants ont été utilisés dans nos expériences afin d'évaluer les méthodologies proposées : CPLEX [11] pour la formulation QIP proposée en 3.1 ainsi que pour les deux formulations pseudo-bouliennes linéaires introduites en 3.3 ; MaxHS [21], WMaxCDCL [14], et OpenWBO [18] pour les deux formulations MaxSAT décrites en 3.4. Nous notons que ces trois solveurs MaxSAT représentent respectivement des approches basées sur la programmation linéaire en nombres entiers (ILP), sur la séparation et évaluation (Branch-and-Bound), et sur les appels aux oracles SAT. Pour réaliser les encodages CNF des contraintes de cardinalité et pseudo-bouliennes, nous avons utilisé la bibliothèque PySAT³ [12, 13], avec les contraintes de cardinalité encodées à l'aide de l'encodage *Cardinality Network* [3], qui permet d'encoder une contrainte de cardinalité de borne $\mathcal{K}$ sur $\mathcal{N}$ littéraux avec une complexité $O(\mathcal{N} \log^2 \mathcal{K})$ en termes de clauses et de variables auxiliaires. Concernant les contraintes pseudo-bouliennes, nous avons utilisé le mode "Best" proposé par PySAT, qui sélectionne automatiquement la méthode d'encodage la plus adaptée.

Pour justifier la non-trivialité des solutions obtenues, nous avons également mis en place une approche naïve par énumération, basée sur le solveur SAT CaDiCaL [5]. Le solveur est exécuté de manière itérative k fois. Après chaque

3. <https://pysathq.github.io/>

itération, une nouvelle clause est ajoutée à la formule pour bloquer l'assignation satisfaisante nouvellement trouvée. Une fois les k itérations terminées, le programme est suspendu et la diversité est calculée à partir des k modèles obtenus.

4.2 Performances globales

Les analyses de performance des différentes approches testées à travers l'ensemble des familles, pour les valeurs de k égales à 2, 5 et 10, sont présentées dans les Tableaux 1, 2 et 3. Ces tableaux présentent les performances sous deux aspects : la capacité des approches proposées à résoudre le problème à l'optimalité et la diversité obtenue par celles-ci, comparée à l'énumérateur naïf CaDiCaL. À partir de ces tableaux, il ressort que toutes les formulations proposées parviennent à obtenir des solutions optimales dans certaines familles, notamment pour des valeurs plus petites de k , comme celle de 2. De manière générale, les formulations linéaires avec les fonctions de poids DW et IW , obtiennent le plus grand nombre de solutions optimales pour toutes les valeurs de k . L'écart se renforce lorsque la valeur de k augmente jusqu'à 10, illustrant ainsi une bonne scalabilité de l'approche linéaire pour la résolution du problème.

Pour les solveurs MaxSAT, OpenWBO coopérant avec DW obtient le plus grand nombre d'instances résolues à l'optimalité, soit 89 sur 108, lorsque $k = 2$, tandis que MaxHS coopérant avec IW bascule en première position lorsque $k = 5$ et $k = 10$, résolvant respectivement 67 et 33 instances. WMaxCDCL présente des performances comparables à celles de MaxHS et OpenWBO lorsque $k = 2$ et $k = 10$, bien que ses performances diffèrent des deux autres lorsque $k = 10$.

De plus, on note que, dans plusieurs cas, bien que la formulation QIP avec CPLEX parvienne à atteindre la valeur optimale prouvée par certains autres solveurs, elle échoue à prouver l'optimalité, avec respectivement 7, 21, et 15 instances non prouvées lorsqu'on prend $k = 2$, $k = 5$ et $k = 10$. Dans l'ensemble, ces résultats démontrent que, malgré le fait que toutes les approches proposées sont capables de résoudre de manière optimale le problème de satisfiabilité diversifiée, le choix du solveur et de la fonction de poids a un impact significatif sur les performances, certaines combinaisons montrant des résultats supérieurs en fonction des cas spécifiques.

En termes de diversité, pour $k = 2, 5, 10$, les meilleurs résultats obtenus par un solveur exact est respectivement de 3.58, 2.50, et 3.34 fois meilleure que celle de la solution obtenue par l'énumération naïve avec CaDiCaL. Lorsqu'on restreint l'analyse aux cas avec optimalité prouvée, ces facteurs deviennent respectivement de 113, 63, et 23. L'écart de performance substantiel entre l'énumérateur naïf et les approches exactes proposées démontre clairement la difficulté inhérente du problème, et confirme la nécessité de développer des approches exactes performantes.

On peut également observer qu'avec l'augmentation de la valeur de k , les approches exactes proposées échouent parfois à obtenir des solutions faisables dans le temps imparti. Cela est particulièrement le cas pour WMaxCDCL

et CPLEX et pourrait être dû au fait que l'objectif interne des solveurs exacts est d'obtenir des solutions optimales, de sorte que les résultats intermédiaires incomplets ne reflètent pas pleinement le processus de résolution. Par exemple, l'approche linéaire avec IW résolue avec le solveur CPLEX ne parvient à obtenir que 4 résultats faisables pour la famille difficile "hardware" lorsque $k = 5$, mais tous ces résultats sont optimaux. Malgré cette difficulté, les approches proposées offrent toujours une qualité satisfaisante pour les solutions sous-optimales, ce qui démontre une grande robustesse.

En se focalisant sur la comparaison de la capacité de résolution optimale de toutes les approches exactes proposées, les Figures 6, 7, 8 offrent une vision plus complète du nombre cumulé d'instances résolues par chaque approche dans le temps imparti.

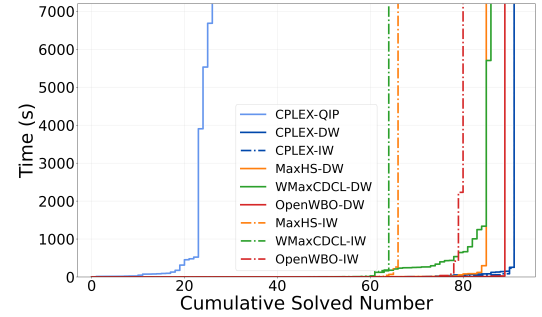


FIGURE 6 – Nombre cumulé d'instances résolues pour $k = 2$ en fonction du temps de résolution en secondes.

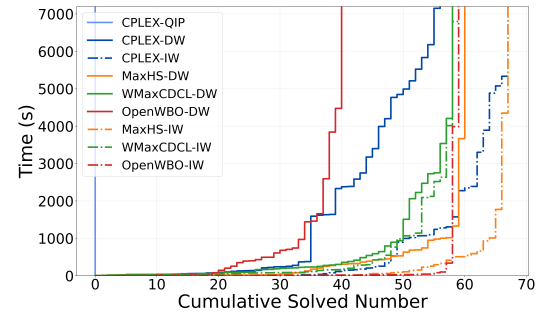


FIGURE 7 – Nombre cumulé d'instances résolues pour $k = 5$ en fonction du temps de résolution en secondes.

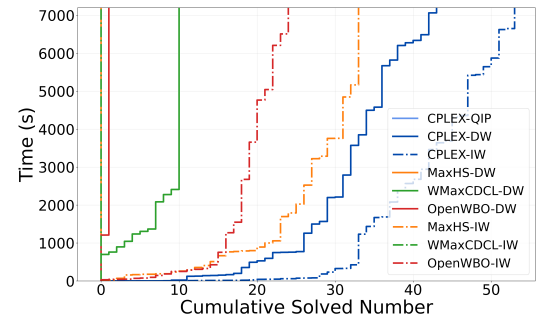


FIGURE 8 – Nombre cumulé d'instances résolues pour $k = 10$ en fonction du temps de résolution en secondes.

Famille	#I	#Var	#CL	Énum. CaDi.	CPLEX			MaxSAT (DW)			MaxSAT (IW)		
					QIP	DW	IW	MaxHS	WMaxCDCL	OpenWBO	MaxHS	WMaxCDCL	OpenWBO
flat100-	30	155	562	[2]	23(6686)[73]	30(11)[103]	30(8)[103]	30(2)[103]	30(15)[103]	30(2)[103]	30(5)[103]	30(16)[103]	30(6)[103]
flat100+	30	425	1584	[2]	1(6685)[63]	30(1047)[283]	30(730)[283]	30(486)[283]	30(357)[283]	30(50)[283]	30(81)[283]	30(435)[283]	25(51)[235]
ais	4	155	2730	[18]	1(4)[6]	4(196)[34]	4(60)[34]	3(56)[34]	4(583)[34]	4(35)[34]	3(69)[34]	3(17)[22]	3(425)[22]
morphed	20	500	3100	[2]	0(-)[0]	20(65)[200]	20(25)[200]	19(166)[200]	20(9836)[200]	19(21)[199]	1(8)[200]	0(-)[0]	18(46)[197]
logistics	4	1881	11682	[3]	1(5531)[216]	3(58)[365]	3(108)[368]	3(286)[364]	2(443)[218]	2(13)[293]	2(304)[366]	1(28)[218]	2(2302)[285]
hardware	20	49391	161548	[5483]	0(-)[596]	4(249)[1177]	4(195)[1180]	0(-)[18835]	0(-)[0]	4(66)[10052]	0(-)[18381]	0(-)[0]	2(74)[9149]
All	108	9476	31620	[1018]	26(18905)[156]	91(1626)[377]	91(1126)[378]	85(996)[3647]	86(16482)[154]	89(187)[2018]	66(466)[3563]	64(651)[116]	80(2904)[1836]

TABLE 1 – Résultats pour $k = 2$. Pour chaque famille d’instances, le nombre d’instances ($\#I$), le nombre moyen de variables ($\#Var$) et le nombre moyen de clauses ($\#CL$) sont indiqués. Ensuite, la diversité des résultats obtenue par l’approche d’énumération naïve utilisant CaDiCaL (CaDi.) est rapportée. Les performances de l’approche QIP et de l’approche linéaire avec CPLEX ainsi que les trois solveurs MaxSAT, à savoir MaxHS, WMaxCDCL et OpenWBO, sur les formulations avec deux fonctions de poids distinctes (DW, IW) sont présentées sous la forme de $\#OPT(tot_time_{opt})[Div_{all}]$, où $\#OPT$ représente le nombre de solutions optimales obtenues dans le temps limite, tot_time_{opt} est la somme des temps de résolution pour ces solutions optimales et Div_{all} est la diversité moyenne calculée en fonction des meilleures solutions faisables (ou optimales) obtenues pour toutes les instances dans la famille concernée. Dans le cas où aucune solution faisable n’est trouvée dans le temps limite, la diversité est considérée comme étant nulle (égale à 0 comme pénalité) sinon la diversité de la meilleure solution faisable est considérée. Parmi toutes nos approches, celle ayant résolu de manière optimale le plus grand nombre d’instances dans le temps le plus court est indiquée en gras. De plus, la meilleure diversité globale Div_{all} parmi toutes les approches, y compris l’énumération naïve avec CaDiCaL, est soulignée.

Famille	#I	#Var	#CL	Énum. CaDi.	CPLEX			MaxSAT (DW)			MaxSAT (IW)		
					QIP	DW	IW	MaxHS	WMaxCDCL	OpenWBO	MaxHS	WMaxCDCL	OpenWBO
flat100-	30	155	562	[50]	0(-)[595]	30(2169)[827]	30(81)[827]	30(3558)[827]	30(5311)[827]	20(7597)[817]	30(609)[827]	30(4472)[827]	28(403)[823]
flat100+	30	425	1584	[32]	0(-)[59]	22(56712)[1493]	29(26379)[2160]	29(13603)[2249]	27(25914)[1973]	20(15181)[2148]	30(3023)[2267]	28(25118)[2067]	30(796)[2267]
ais	4	155	2730	[280]	0(-)[50]	1(5834)[119]	1(2348)[124]	1(429)[314]	1(645)[124]	0(-)[304]	1(506)[327]	1(765)[124]	1(3987)[306]
morphed	20	500	3100	[30]	0(-)[87]	0(-)[1977]	3(10458)[1985]	0(-)[1855]	0(-)[0]	0(-)[1197]	6(9311)[2000]	0(-)[0]	0(-)[2000]
logistics	4	1881	11682	[38]	0(-)[0]	0(-)[995]	0(-)[1608]	0(-)[2604]	0(-)[0]	0(-)[1682]	0(-)[2652]	0(-)[0]	0(-)[1782]
hardware	20	49391	161548	[42853]	0(-)[1919]	3(16991)[3850]	4(4427)[3850]	0(-)[82752]	0(-)[0]	0(-)[87145]	0(-)[100503]	0(-)[0]	0(-)[8613]
All	108	9476	31620	[7976]	0(-)[555]	56(81706)[1765]	67(43693)[1974]	60(17589)[16630]	58(31870)[782]	40(22779)[17257]	67(13449)[19952]	59(30354)[808]	59(5185)[2901]

TABLE 2 – Résultats pour $k = 5$. Le format des données est aligné avec Tableau 1.

Famille	#I	#Var	#CL	Énum. CaDi.	CPLEX			MaxSAT (DW)			MaxSAT (IW)		
					QIP	DW	IW	MaxHS	WMaxCDCL	OpenWBO	MaxHS	WMaxCDCL	OpenWBO
flat100-	30	155	562	[337]	0(-)[1501]	30(14651)[3410]	30(1253)[3410]	0(-)[3224]	10(14085)[1815]	1(1209)[2897]	27(21543)[3410]	0(-)[2144]	17(31749)[3410]
flat100+	30	425	1584	[179]	0(-)[20]	5(18333)[1487]	16(43420)[3905]	0(-)[8323]	0(-)[0]	0(-)[8286]	6(20519)[9350]	0(-)[1531]	7(3270)[9350]
ais	4	155	2730	[1226]	0(-)[0]	0(-)[211]	0(-)[211]	0(-)[1272]	0(-)[0]	0(-)[1230]	0(-)[1366]	0(-)[0]	0(-)[1278]
morphed	20	500	3100	[168]	0(-)[0]	8(45029)[7988]	7(31263)[7993]	0(-)[6548]	0(-)[0]	0(-)[5106]	0(-)[6371]	0(-)[0]	0(-)[8000]
logistics	4	1881	11682	[208]	0(-)[0]	0(-)[0]	0(-)[1942]	0(-)[9230]	0(-)[0]	0(-)[7772]	0(-)[9975]	0(-)[0]	0(-)[7760]
hardware	20	49391	161548	[105280]	0(-)[0]	0(-)[14790]	0(-)[15886]	0(-)[330172]	0(-)[0]	0(-)[34762]	0(-)[42646]	0(-)[0]	0(-)[21896]
All	108	9476	31620	[19724]	0(-)[423]	43(78013)[5586]	53(75936)[6534]	0(-)[65952]	10(14085)[504]	1(1209)[10823]	33(42062)[13042]	0(-)[1021]	24(35019)[9416]

TABLE 3 – Résultats pour $k = 10$. Le format des données est aligné avec Tableau 1.

Lorsque $k = 2$, DW surpasse généralement IW , à l'exception du seul cas où CPLEX- DW et CPLEX- IW parviennent à résoudre rapidement toutes les instances. À mesure que k augmente à 5 et 10, IW montre généralement une meilleure performance que le poids direct. Par ailleurs, on observe que la formulation quadratique (QIP) montre des performances plus faibles par rapport aux approches linéaires. De plus, l'approche linéaire avec CPLEX se comporte de manière comparable aux solveurs MaxSAT uniquement lorsque $k = 5$, mais elle présente un avantage dans tous les autres cas. Ces résultats confirment que nos approches exactes peuvent résoudre efficacement les problèmes de satisfiabilité diversifiée de taille raisonnable, les approches linéaires montrant des résultats particulièrement prometteurs.

4.3 Poids direct vs poids incrémental

Pour analyser les différences entre les deux fonctions de poids proposées, DW et IW , nous menons une comparaison détaillée de leurs performances à travers différentes familles d'instances et configurations de solveurs. Les Figures 9 et 10 présentent les comparaisons des temps d'exécution des deux fonctions de poids pour $k = 2$ et $k = 5$. Chaque point dans les diagrammes de dispersion correspond à une instance, avec ses coordonnées déterminées par le temps d'exécution pour les deux fonctions de poids. Lorsqu'une instance n'a pas été résolue dans le temps imparti, la limite du temps d'exécution est reportée. Les points situés en dessous de la diagonale indiquent les instances où IW réalise de meilleures performances, tandis que les points au-dessus de la diagonale sont en faveur de DW .

Nous observons que les distributions des points suivent des tendances distinctes lorsque la valeur de k varie. Lorsque $k = 2$, DW montre généralement des performances supérieures, avec 33.33%, 62.96%, 95.37%, 92.59% des points situés au-dessus ou sur la ligne diagonale respectivement pour les solveurs CPLEX, MaxHS, WMaxCDCL, OpenWBO. À mesure que k augmente à 5, nous observons une tendance où IW surpasse DW , avec des pourcentages de points de 84.26%, 89.81%, 81.48%, 72.22% situés sous ou sur la ligne diagonale respectivement pour les solveurs CPLEX, MaxHS, WMaxCDCL et OpenWBO. Dans le cas de $k = 10$, nous omettons les figures correspondantes car toutes les approches, à l'exception des modèles linéaires avec CPLEX, n'ont pas réussi à résoudre de nombreuses instances, mais nous observons clairement dans le Tableau 3 que IW obtient les meilleurs résultats. Enfin, sur la base des résultats obtenus pour toutes les valeurs de k , nous constatons l'avantage de IW par rapport à DW à mesure que k augmente.

5 Conclusion

Dans cette étude, nous avons proposé des formulations exactes pour le problème de la satisfiabilité diversifiée, en utilisant la programmation entière quadratique (QIP) ainsi que des modèles de programmation linéaire pseudo-boulienne, pouvant être adaptées en formulations MaxSAT. Ces approches permettent de formuler l'objectif de la satis-

fiabilité diversifiée sous l'angle de la diversité des variables. L'approche QIP présente une formulation naturelle de l'objectif sous forme quadratique, tandis que les formulations linéaires et MaxSAT reposent sur deux fonctions de poids distinctes, exploitant pleinement les capacités des solveurs modernes à résoudre des problèmes d'optimisation linéaire complexes. Les résultats expérimentaux, menés sur un large éventail d'instances avec différentes valeurs du paramètre k , représentant le nombre de modèles, démontrent l'efficacité et la robustesse des méthodologies proposées.

Dans le cadre de nos travaux futurs, nous envisageons de concevoir des algorithmes dédiés pour la satisfiabilité diversifiée qui dépassent les limitations des schémas de résolution globaux tout en conservant la structure et la représentation de l'objectif proposées dans nos approches actuelles. Par ailleurs, à mesure que nous élargissons nos expériences à un éventail d'instances plus large, nous espérons établir de nouvelles solutions optimales, afin de soutenir les chercheurs intéressés par ce domaine. Ces repères élargis faciliteront également la conception et l'évaluation d'algorithmes heuristiques et d'approximation pour le problème de la satisfiabilité diversifiée.

Remerciements

Ce travail est partiellement soutenu par le projet ANR-24-CE23-6126 (BforSAT) et par la chaire IA ANR-19-CHIA-0013-01 (MASSAL'IA) co-financée par l'agence nationale de recherche et le gestionnaire du réseau de distribution d'électricité Enedis. Il a bénéficié d'un accès aux ressources HPC de la « Plateforme MatriCS » de l'Université de Picardie Jules Verne qui est cofinancée par l'Union Européenne avec le Fonds Européen de Développement Régional (FEDER) et le Conseil Régional des Hauts-De-France entre autres.

Références

- [1] S. Agbaria, D. Carmi, O. Cohen, D. Korchemny, M. Lifshits, and A. Nadel. Sat-based semiformal verification of hardware. In R. Bloem and N. Sharygina, editors, *FMCAD 2010*, pages 25–32. IEEE, 2010.
- [2] O. Angelsmark and J. Thapper. Algorithms for the maximum hamming distance problem. In B. Faltings, A. Petcu, F. Fages, and F. Rossi, editors, *CSCLP 2004*, volume 3419 of *LNCS*, pages 128–141. Springer, 2004.
- [3] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks and their applications. In O. Kullmann, editor, *SAT 2009*, volume 5584 of *LNCS*, pages 167–180. Springer, 2009.
- [4] F. Bacchus, M. Jarvisalo, and R. Martins. Maximum satisfiability. In *Handbook of satisfiability*, pages 929–991. IOS Press, 2021.
- [5] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In T. Balyo, N. Froylenks, M. Heule, M. Iser, M. Jarvisalo, and M. Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

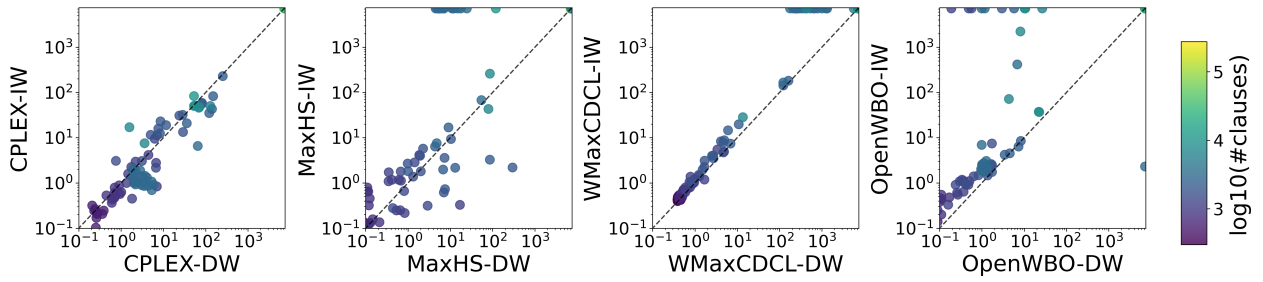


FIGURE 9 – Comparaison des temps d’exécution en échelle logarithmique des deux fonctions de poids pour $k = 2$.

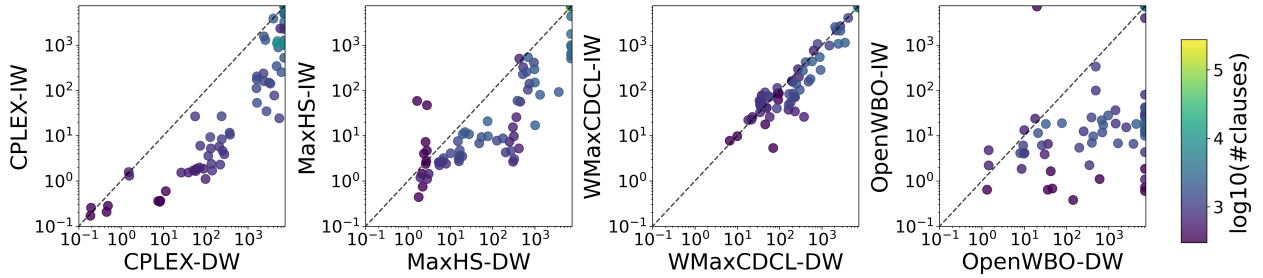


FIGURE 10 – Comparaison des temps d’exécution en échelle logarithmique des deux fonctions de poids pour $k = 5$.

- [6] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.
- [7] E. M. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In K. Jensen and A. Podolski, editors, *TACAS 2004*, volume 2988 of *LNCS*, pages 168–176. Springer, 2004.
- [8] S. A. Cook. The complexity of theorem-proving procedures. In M. A. Harrison, R. B. Banerji, and J. D. Ullman, editors, *STOC 1971*, pages 151–158. ACM, 1971.
- [9] P. Crescenzi and G. Rossi. On the hamming distance of constraint satisfaction problems. *Theor. Comput. Sci.*, 288(1):85–100, 2002.
- [10] P. Hammer Ivănescu and S. Rudeanu. Boolean methods in operations research and related areas. *Oekonometrie und Unternehmensforschung* (, 7, 1968.
- [11] IBM Corporation. *IBM ILOG CPLEX Optimization Studio*, 2022.
- [12] A. Ignatiev, A. Morgado, and J. Marques-Silva. PySAT : A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.
- [13] A. Ignatiev, Z. L. Tan, and C. Karamanos. Towards universally accessible SAT technology. In *SAT*, pages 4 :1–4 :11, 2024.
- [14] C. Li, Z. Xu, J. Coll, F. Manyà, D. Habet, and K. He. Boosting branch-and-bound maxsat solvers with clause learning. *AI Commun.*, 35(2):131–151, 2022.
- [15] C. M. Li and F. Manyà. MaxSAT, Hard and Soft Constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 903–927. IOS Press, 2021.
- [16] I. Lynce and J. Marques-Silva. SAT in bioinformatics : Making the case with haplotype inference. In A. Biere and C. P. Gomes, editors, *SAT 2006*, volume 4121 of *LNCS*, pages 136–141. Springer, 2006.
- [17] P. Manolios, M. G. Oms, and S. O. Valls. Checking pedigree consistency with PCS. In O. Grumberg and M. Huth, editors, *TACAS 2007*, volume 4424 of *LNCS*, pages 339–342. Springer, 2007.
- [18] R. Martins, V. Manquinho, and I. Lynce. Open-wbo : A modular maxsat solver,. In C. Sinz and U. Egly, editors, *SAT 2014*, volume 8561 of *LNCS*, pages 438–445. Springer, 2014.
- [19] N. Misra, H. Mittal, and A. Rai. On the parameterized complexity of diverse sat. *arXiv preprint arXiv :2412.09717*, 2024.
- [20] A. Nadel. Generating diverse solutions in SAT. In K. A. Sakallah and L. Simon, editors, *SAT 2011*, volume 6695 of *LNCS*, pages 287–301. Springer, 2011.
- [21] A. Niskanen, J. Berg, and M. Järvisalo. Maxhs in maxsat evaluation 2022. *MaxSAT Evaluation 2022*, page 35, 2022.
- [22] A. D. Pia, S. S. Dey, and M. Molinaro. Mixed-integer quadratic programming is in np. *Mathematical Programming*, 162 :225–240, 2017.
- [23] O. Roussel and V. Manquinho. Pseudo-boolean and cardinality constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1087–1129. IOS Press, 2021.
- [24] P. Surynek. Lazy compilation of variants of multi-robot path planning with satisfiability modulo theory (SMT) approach. In *IROS 2019*, pages 3282–3287. IEEE, 2019.
- [25] H. Xu, R. A. Rutenbar, and K. A. Sakallah. sub-sat : a formulation for relaxed boolean satisfiability with applications in routing. In S. S. Sapatnekar and M. Pedram, editors, *ISPD 2002*, pages 182–187. ACM, 2002.

Prétraitement PLNE pour la simplification des instances MaxSAT

Jialu Zhang¹, Chu-Min Li¹, Sami Cherif¹, Shuolin Li², Zhifei Zheng¹

¹ Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

² Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

{jialu.zhang;chu-min.li;sami.cherif;zhifei.zheng}@u-picardie.fr;shuolin.li@lis-lab.fr

Résumé

Le problème de satisfiabilité maximum (MaxSAT) constitue un défi d'optimisation majeur avec de nombreuses applications du monde réel. Cet article propose une approche qui intègre des techniques de prétraitement issues de la programmation linéaire en nombres entiers (PLNE) dans les solveurs MaxSAT. Les résultats expérimentaux montrent que ce prétraitement améliore l'efficacité des solveurs MaxSAT modernes sur les récents benchmarks de l'évaluation MaxSAT. Une analyse approfondie révèle que cette méthode peut réduire le nombre de variables d'environ 40% et le nombre de clauses d'environ 60% pour certaines instances MaxSAT. En particulier, elle est particulièrement efficace sur la famille d'instances « extension enforcement », permettant au solveur WMaxCDCL de réduire le temps CPU de plus de 65% lors de la résolution de ces cas.

Mots-clés

Satisfiabilité Maximum, PLNE, prétraitement

Abstract

The Maximum Satisfiability problem (MaxSAT) is a major optimization challenge with numerous practical applications. This paper proposes an approach that integrates preprocessing techniques from Integer Linear Programming (ILP) into MaxSAT solvers. Experimental results show that this preprocessing enhances the efficiency of modern MaxSAT solvers on recent MaxSAT evaluation benchmarks. A detailed analysis reveals that this method can reduce the number of variables by approximately 40% and the number of clauses by around 60% for certain MaxSAT instances. Notably, it proves particularly effective for the "extension enforcement" instance family, enabling the WMaxCDCL solver to reduce CPU time by more than 65% when solving these instances.

Keywords

Maximum Satisfiability, ILP, Preprocessing.

1 Introduction

La satisfiabilité maximum (MaxSAT) constitue une extension naturelle de la satisfiabilité propositionnelle (SAT) en problème d'optimisation [4]. Pour une formule donnée en forme normale conjonctive (CNF), SAT consiste

à vérifier s'il existe une interprétation des variables qui satisfait toutes les contraintes clausales de la formule. Dans MaxSAT, l'objectif est de trouver le nombre maximum de clauses pouvant être satisfaites dans la formule. De nombreux problèmes d'optimisation du monde réel peuvent être modélisés sous forme d'instances MaxSAT, notamment dans les domaines de la planification et de l'ordonnancement [5, 6, 15] ou encore de la vérification matérielle et logicielle [12], entre autres. Malheureusement, MaxSAT est plus difficile à résoudre que SAT, tant en théorie qu'en pratique, car il est plus complexe de trouver et de prouver l'optimalité des solutions [7].

Les algorithmes de résolution du problème MaxSAT peuvent être globalement classés en deux catégories : les algorithmes exacts et les algorithmes heuristiques. Les algorithmes exacts, tels que ceux de type séparation (BnB) et évaluation et la programmation linéaire en nombres entiers (PLNE), visent à trouver la solution optimale tout en démontrant son optimalité. En revanche, les algorithmes heuristiques, comme la recherche locale ou le recuit simulé, sont généralement plus rapides, mais ne garantissent pas l'optimalité [7, 8]. Il est reconnu que les solveurs PLNE, bien qu'efficaces sur certaines familles d'instances, ne sont pas compétitifs pour la majorité des instances MaxSAT (industrielles ou aléatoires) [2]. Ainsi, la pratique courante observée dans les évaluations MaxSAT¹ récentes, en particulier pour les solveurs les plus performants, consiste à combiner les solveurs PLNE dans un portfolio avec d'autres types de solveurs.

À titre d'exemple, dans la dernière évaluation MaxSAT [3], où la limite de temps totale est de 3600 secondes par instance, le solveur EvalMaxSAT exécute d'abord le solveur PLNE SCIP [1] pendant 400 secondes, puis un solveur basé sur l'algorithme OLL [9] pendant 3200 secondes. De même, UWMaxSat [11] alterne l'exécution de SCIP et de son propre algorithme MaxSAT avec des limites de temps possiblement différentes, en comparant les bornes supérieures et inférieures obtenues pour les améliorer mutuellement. Ainsi, dans ces solveurs hybrides tirant parti de PLNE, le solveur PLNE est généralement utilisé de manière indépendante dans un cadre de portfolio, nécessitant un réglage heuristique minutieux, notamment sur la gestion des durées d'exécution.

¹<https://maxsat-evaluations.github.io/>

Dans cet article, contrairement aux méthodes existantes, nous proposons une approche unifiée dans laquelle la PLNE est entièrement intégrée au processus de résolution en tant qu'étape de prétraitement. Notre approche consiste d'abord à lire la formule CNF pour la convertir en un ensemble de contraintes linéaires entières avec une fonction objectif. Ensuite, un solveur PLNE est appelé pour simplifier le problème, avant que les contraintes linéaires simplifiées ne soient retransformées en forme clausale afin d'être résolues par un solveur MaxSAT. Les résultats expérimentaux montrent que cette approche permet de résoudre un plus grand nombre d'instances que les solveurs MaxSAT de l'état de l'art. De plus, notre méthode ne nécessite pas de définir des limites de temps heuristiques pour les solveurs PLNE, qui continuent à s'exécuter jusqu'à la fin du prétraitement, contrairement à d'autres approches telles que EvalMaxSAT ou UWrMaxSat.

La suite de ce papier est organisée comme suit. La Section 2 introduit formellement le problème MaxSAT et présente brièvement les techniques de prétraitement en PLNE. La Section 3 décrit notre méthode d'intégration des techniques de prétraitement PLNE dans un solveur MaxSAT. La Section 4 présente les résultats expérimentaux. Enfin, on conclut et on discute les perspectives de nos travaux dans la Section 5.

2 Préliminaires

2.1 Satisfiabilité Maximum

Étant donné un ensemble de variables propositionnelles $V = \{x_1, x_2, \dots, x_n\}$, un littéral l est soit une variable x , soit sa négation $\neg x$. Une clause c est une disjonction de k littéraux $l_1 \vee \dots \vee l_k$, et peut être représentée comme un ensemble de littéraux. Une formule F en forme normale conjonctive (CNF) est une conjonction de m clauses $c_1 \wedge \dots \wedge c_m$. Une variable x est dite affectée si elle prend la valeur 0 (*false*) ou 1 (*true*). Un littéral x est évalué à vrai (faux) si la variable x est affectée à 1 (0). Pour $\neg x$, c'est l'inverse. Une clause c est satisfaite si l'un de ses k littéraux est évalué à vrai. Une formule F est satisfaite uniquement lorsque toutes ses clauses le sont. Une affectation est dite *complète* si toutes les variables sont affectées, sinon elle est dite *partielle*. Le problème de satisfiabilité propositionnelle (SAT) consiste à trouver une affectation complète qui satisfait une formule CNF donnée [4].

MaxSAT est une extension naturelle du problème SAT en problème d'optimisation, et consiste à trouver une affectation qui maximise le nombre de clause satisfaites dans une formule CNF donnée. Dans la suite, on emploie le terme *MaxSAT* pour désigner la version la plus générique de ce problème: MaxSAT partiel pondéré [7]. MaxSAT partiel divise les clauses en deux catégories : les clauses *dures* et les clauses *souples*, c'est-à-dire $F = H \cup S$. L'objectif est de trouver une affectation qui satisfait toutes les clauses dures H tout en maximisant le nombre de clauses souples satisfaites dans S . Dans MaxSAT partiel pondéré, chaque clause souple $c \in S$ est associée à un poids entier positif w_c . L'objectif consiste alors à trouver une affectation optimale

qui maximise la somme des poids des clauses souples satisfaites dans S , tout en satisfaisant l'ensemble des clauses dures dans H .

2.2 Techniques de prétraitement

Les solveurs PLNE utilisent des techniques de prétraitement afin de transformer un modèle donné en un modèle équivalent, mais potentiellement plus facile à résoudre. Ces techniques incluent la fixation de variables, l'agrégation de variables, l'élimination de contraintes redondantes, ainsi que d'autres mécanismes d'inférence avancés [13]. La technique de fixation de variables repose sur un algorithme de *probing* qui attribue temporairement la valeur 0 ou 1 à une variable binaire, puis propage les implications qui en découlent. L'agrégation de variables exploite les équations et les relations de contraintes présentes dans le modèle, ainsi que des algorithmes de détection de clusters, afin de regrouper plusieurs variables en une seule. Par ailleurs, l'élimination des contraintes redondantes permet de supprimer les contraintes qui sont implicitement satisfaites par d'autres, simplifiant ainsi le modèle [1].

Des techniques de prétraitement sont également utilisées dans les solveurs SAT et MaxSAT, telles que la propagation unitaire, la détection de *failed literals*, la résolution auto-subsomante, ainsi que d'autres méthodes avancées [8]. Bien que ces techniques partagent des similarités conceptuelles avec celles employées en PLNE, leur implémentation diffère. Afin de déterminer si les solveurs MaxSAT peuvent bénéficier des techniques de prétraitement issues de PLNE, il est nécessaire de convertir un problème MaxSAT en un problème PLNE. Un tel modèle pour MaxSAT partiel pondéré est donné par l'équation (1). Soit c une clause dans S (resp. H) de la forme $S_c^- \vee S_c^+$ (resp. $H_c^- \vee H_c^+$), où S_c^- (resp. S_c^+) est une disjonction de littéraux négatifs (resp. positifs) tels que $\neg x$ (resp. x). De nouvelles variables binaires y_x sont introduites pour chaque variable propositionnelle x , et z_c pour chaque clause souple c de la formule.

$$\textbf{Objectif:} \quad \text{Maximiser} \sum_{c \in S} w_c \cdot z_c \quad (1a)$$

Sous contraintes:

$$\sum_{x \in H_c^+} y_x + \sum_{x \in H_c^-} (1 - y_x) \geq 1, \quad \forall c \in H \quad (1b)$$

$$z_c \leq \sum_{x \in S_c^+} y_x + \sum_{x \in S_c^-} (1 - y_x), \quad \forall c \in S \quad (1c)$$

$$z_c \in \{0, 1\}, \quad \forall c \in S \quad (1d)$$

$$y_x \in \{0, 1\}, \quad \forall x \in V \quad (1e)$$

3 Prétraitement PLNE pour MaxSAT

Cette section présente notre méthodologie visant à intégrer les techniques de prétraitement PLNE dans les solveurs MaxSAT, ainsi que les encodages des variables et des contraintes.

3.1 Méthodologie

Pour intégrer les techniques de prétraitement de la PLNE dans un solveur MaxSAT, nous proposons la méthodologie suivante :

1. À partir d'une instance MaxSAT (*originInst*), construire le modèle PLNE (*originModel*) en utilisant les équations (1a)–(1e).
2. Appliquer des techniques de prétraitement à *originModel* à l'aide d'un solveur PLNE, puis extraire le modèle prétraité (*preModel*).
3. Convertir *preModel* en une instance MaxSAT simplifiée (*simpInst*). Si la conversion échoue, interrompre le prétraitement et revenir à la résolution de *originInst*.
4. Résoudre *simpInst* à l'aide d'un solveur MaxSAT afin d'obtenir la solution de l'instance simplifiée (*simpSol*).
5. Transformer *simpSol* en une solution de l'instance initiale (*orgSol*) et retourner le résultat.

L'aspect clé de notre méthodologie réside dans la conversion de *preModel* en *simpInst*. Nous commençons par examiner les types de variables et de contraintes présents dans *preModel*, puis on procède à leur encodage au format CNF pondéré. Cet encodage consiste à associer les variables de *preModel* à celles de *simpInst*, à encoder les contraintes sous forme de clauses dures, et à représenter la fonction objectif sous forme de clauses souples. Les détails de cette procédure sont décrits dans les sous-sections suivantes.

3.2 Encodage des variables

Les variables dans *preModel* peuvent être affectées, agrégées ou libres. Nous sauvegardons les valeurs des variables affectées afin de faciliter la reconstruction de la solution initiale. Pour les variables non affectées (sans valeur fixée) de *preModel*, nous appliquons l'algorithme 1 afin de déterminer leurs littéraux correspondants dans *simpInst*. Pour simplifier, nous considérons uniquement deux types d'opérations d'agrégation : $x = y$ et $x = \neg y$. Si l'agrégation implique plus de deux variables, nous retournons un échec et interrompons le processus de prétraitement. La structure de données d'entrée, *vpool*, maintient la correspondance entre les variables de *preModel* et leurs équivalents dans *simpInst*, garantissant ainsi une reconstruction correcte de la solution de l'instance originale.

Exemple 1. On suppose qu'on a trois variables dans *preModel* avec les relations d'agrégation suivantes : $(x = \neg y)$ et $(y = z)$. Il suffit alors de créer une nouvelle variable v_1 dans *simpInst* pour remplacer x , y et z . La correspondance finale est la suivante : $\{x \rightarrow \neg v_1, y \rightarrow v_1, z \rightarrow v_1\}$, ce qui garantit les équivalences $(x = \neg y)$ et $(y = z)$.

Input : Variable x , table de correspondance *vpool*

Output : Littéral *lit* ou 'échec'

```

1 if  $x$  n'est pas agrégée then
2   if  $x$  n'est pas dans vpool then
3      $vpool[x] \leftarrow$  nouvelle variable dans simpInst;
4   return  $vpool[x]$ ;
5 else if  $x$  est agrégée à  $y$  then
6    $lit \leftarrow GetVarLit(y, vpool)$ ;
7   if  $x = \neg y$  then
8     return  $\neg lit$ ;
9   else
10    return  $lit$ ;
11 else
12   return 'échec';

```

Algorithme 1 : *GetVarLit*($x, vpool$)

3.3 Encodage des contraintes

Nous utilisons le solveur SCIP pour effectuer le prétraitement de *originModel*, car il s'agit d'un solveur open-source de programmation en nombres entiers mixtes largement utilisé dans les évaluations MaxSAT. SCIP propose une interface permettant d'extraire des informations détaillées à partir de *preModel*. Nous prenons en compte les types de contraintes listés ci-dessous, générés par SCIP dans *preModel*. Ces contraintes — en particulier la contrainte non linéaire *Logical AND* — proviennent principalement de techniques d'extraction de portes logiques [10]. D'autres types de contraintes, telles que les contraintes pseudo-bouliennes [4], seront prises en charge dans le cadre de travaux futurs.

- **Contrainte Logical OR:** $\sum_{i=1}^n x_i \geq 1$, encodée par la clause $(x_1 \vee x_2 \vee \dots \vee x_n)$.
- **Contrainte Logical AND:** $\prod_{i=1}^n x_i = y$, encodée par les clauses $(y \vee \neg x_1 \vee \neg x_2 \vee \dots \vee \neg x_n), (\neg y \vee x_i) \forall 1 \leq i \leq n$.
- **Contrainte Setppc packing:** $\sum_{i=1}^n x_i \leq 1$, encodée par les clauses $(\neg x_i \vee \neg x_j) \forall 1 \leq i < j \leq n$.
- **Contrainte Setppc partitioning:** $\sum_{i=1}^n x_i = 1$, encodée par les clauses $(x_1 \vee x_2 \vee \dots \vee x_n), (\neg x_i \vee \neg x_j) \forall 1 \leq i < j \leq n$.

L'algorithme parcourt l'ensemble des contraintes dans *preModel*. Si une contrainte correspond à un type reconnu, elle est encodée sous forme d'une ou plusieurs clauses dures et ajoutée à *simpInst*. Dans le cas contraire, le prétraitement est interrompu et l'instance MaxSAT originale est transmise au solveur MaxSAT. Concernant la fonction objectif, les variables de décision sont encodées sous forme de clauses souples, en leur attribuant des poids correspondant aux coefficients des variables dans le modèle PLNE. Afin de limiter le temps consommé durant le prétraitement, la taille de l'instance d'entrée est d'abord vérifiée. Si elle dépasse un seuil prédéfini, le prétraitement est ignoré. De

MSE19-24	SCIP	CPLEX
1843	809	850

Table 1: Comparaison du nombre d'instances résolues par les solveurs PLNE.

Solveur	Avec prétraitement	Sans prétraitement
RC2	1220	1238
WMaxCDCL	1426	1436
EvalMaxSAT	1433	1435
UWrMaxSat	1443	1444

Table 2: Comparaison des solveurs MaxSAT, avec et sans prétraitement, en fonction du nombre d'instances résolues.

plus, comme la taille de *simpInst* n'est pas nécessairement inférieure à celle de l'instance initiale, nous réutilisons l'instance d'origine dans le cas où *simpInst* s'avère plus volumineuse.

4 Évaluation expérimentale

Nous utilisons des solveurs PLNE et MaxSAT de l'état de l'art pour nos expériences. Plus précisément, les solveurs PLNE utilisés sont SCIP (version 9.1.1) et CPLEX (version 22.1.1.0). Parmi les solveurs MaxSAT, le solveur RC2 a remporté l'évaluation MaxSAT de 2019, tandis que les solveurs WMaxCDCL, UWrMaxSat et EvalMaxSAT sont classés parmi les trois meilleurs dans la catégorie non pondérée de l'édition 2024. Les instances MaxSAT proviennent de la catégorie non pondérée des évaluations MaxSAT de 2019 à 2024 (MSE19-24). Afin de garantir l'équité des tests, les doublons ont été supprimés. Les calculs ont été réalisés sur la plateforme Matrics², équipée d'un processeur AMD EPYC 7502 à 64 cœurs (2,5 GHz) et de 1000 Go de mémoire vive. Chaque instance de test est exécutée sur un cœur, avec 31 Go de RAM et une limite de temps fixée à 3600 secondes.

Nous avons comparé notre méthodologie aux solveurs PLNE ainsi qu'aux solveurs MaxSAT. La Table 1, la Table 2 et la Figure 1 présentent, pour chaque solveur, le nombre d'instances résolues ainsi que les temps de calcul correspondants. D'après les résultats, bien que les solveurs PLNE obtiennent des performances modestes sur l'ensemble du benchmark, leurs techniques de prétraitement permettent de simplifier les instances originales. Cette simplification a permis aux solveurs WMaxCDCL et RC2 de résoudre respectivement 10 et 18 instances supplémentaires. En revanche, les techniques de prétraitement n'améliorent pas de manière significative les performances des solveurs EvalMaxSAT et UWrMaxSat, probablement en raison de l'intégration déjà existante de SCIP dans ces deux solveurs [3].

Nous avons ensuite réalisé des expériences supplémentaires sur le solveur WMaxCDCL. La Table 3 présente le nombre d'instances affectées par le processus de prétraitement. Les

Famille	Instances	+Petites	+Grandes	Échouées	Ignorées
MS19only	427	84	148	129	66
MS20only	401	54	90	124	133
MS21only	254	42	63	83	66
MS22only	254	33	110	65	46
MS23only	260	37	73	100	50
MS24only	247	19	62	114	52
Total	1843	269	546	615	413

Table 3: Nombre d'instances plus petites (+Petites), plus grandes (+Grandes), échouées et Ignorées pour chaque famille du benchmark. Les instances ignorées sont celles contenant plus de 200 000 variables ou 1 000 000 de clauses.

Métriques (moy)	+Petites	+Grandes
temps de prétraitement	2.88s	19.71s
Taux de variables affectées	34%	6%
Taux de variables agrégées	36%	23%

Table 4: Temps moyen de prétraitement et taux de variables affectées et agrégées.

résultats démontrent que le prétraitement permet de simplifier avec succès 269 instances. En revanche, 615 instances n'ont pas pu être converties en raison de contraintes non prises en charge ou d'agrégations complexes, et 546 instances ont été prétraitées en devenant plus volumineuses que les originales. Cela montre que les techniques de prétraitement PLNE ne sont pas toujours pertinentes pour les instances MaxSAT.

Nous avons également comparé diverses métriques des instances simplifiées avec succès et des instances plus grandes dans la Table 4. D'après les résultats, nous avons observé un contraste marqué entre les deux groupes. Certaines instances sont plus facilement simplifiées par les techniques de prétraitement, tandis que d'autres ne le sont pas. Le taux de variables affectées est nettement plus élevé pour les instances simplifiées avec succès, ce qui indique que la technique de fixation des variables joue un rôle important dans la simplification des instances.

Pour évaluer les bénéfices que WMaxCDCL tire des instances simplifiées, nous avons augmenté la limite de temps à 10 000 secondes et effectué des tests sur les instances de la famille "extension enforcement", qui modélise un problème dans le domaine de l'argumentation abstraite, inclu-

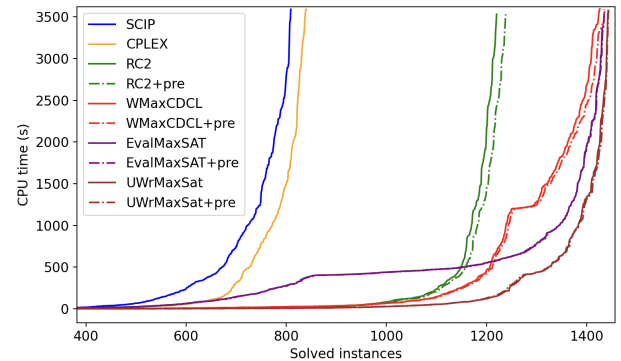


Figure 1: Nombre d'instances résolues vs. temps CPU

²<https://www.matrics.u-picardie.fr>

Inst	Temps (s)			Taux (%)	
	Origin	Simp	Prétraitement	Vars	Clauses
1	670	54	2.7	30.3	57.1
2	1284	379	2.8	21.3	51.5
3	200	106	3.0	33.6	59.1
4	985	270	2.4	43.5	64.6
5	2814	1549	16.0	42.7	64.6
6	3645	1012	15.1	34.4	59.6
7	>10000	5614	16.9	32.3	58.3
8	>10000	4177	18.3	35.9	60.5
9	4786	1252	16.0	38.5	62.1
10	>10000	3136	14.5	34.2	59.5
11	3170	296	14.5	44.4	65.7
12	589	734	16.8	32.3	58.3
13	5181	804	15.7	37.3	61.4
14	2992	1314	15.7	36.8	61.0
15	5000	1328	15.2	37.3	61.3
16	1774	440	10.8	56.2	72.5
17	1115	193	9.3	56.7	72.8

Table 5: Temps CPU et taux de réduction des variables et des clauses sur les instances de la famille "extension enforcement" (no-strict). "Origin" et "Simp" représentent respectivement le temps CPU pris par WMaxCDCL pour résoudre une instance originale et une instance simplifiée.

ant les variantes *strict* et *nonstrict* [14]. Nous avons choisi la variante *nonstrict* de cette famille, car WMaxCDCL a résolu 5 instances supplémentaires grâce aux techniques de prétraitement proposées en 3600 secondes. Les résultats sont présentés dans la Table 5. Le taux de réduction du nombre de variables est calculé selon la formule $(Origin_{vars} - Simp_{vars}) / Origin_{vars} * 100\%$, la même formule étant appliquée au nombre de clauses. D'après les résultats, nous observons que le prétraitement a permis de réduire environ 40% des variables et 60% des clauses, ce qui a permis à WMaxCDCL d'économiser plus de 65% du temps CPU lors de la résolution de ces instances.

5 Conclusion

Dans cet article, nous avons proposé une méthodologie pour intégrer les techniques de prétraitement PLNE dans les solveurs MaxSAT. Les résultats expérimentaux montrent que ces techniques de prétraitement permettent à WMaxCDCL, le vainqueur de l'évaluation MaxSAT 2024 dans la catégorie non pondérée, de résoudre 10 instances supplémentaires dans cette catégorie et de réduire la consommation de temps CPU de plus de 65% pour la résolution des instances de la famille "extension enforcement". Cependant, le prétraitement n'est pas toujours bénéfique pour les solveurs MaxSAT, car il peut parfois rendre les instances simplifiées plus grandes que les originales. De plus, les techniques de prétraitement n'apportent pas d'amélioration substantielle pour les solveurs MaxSAT qui utilisent un solveur PLNE dans un portfolio. Nos travaux futurs viseront à étendre le types de contraintes supportées ainsi que l'agrégation complexe de variables dans *preModel*. Il serait également pertinent de réaliser une analyse approfondie des techniques de prétraitement dans les solveurs PLNE afin d'en améliorer la compatibilité avec les solveurs MaxSAT.

Remerciements

Ce travail est partiellement soutenu par la chaire IA ANR-19-CHIA-0013-01 (MASSAL'IA) co-financée par l'agence nationale de recherche et le gestionnaire du réseau de distribution d'électricité Enedis, et par le projet ANR-24-CE23-6126 (BforSAT). Il a bénéficié d'un accès aux ressources HPC de la « Plateforme MatriCS » de l'Université de Picardie Jules Verne qui est cofinancée par l'Union Européenne avec le Fonds Européen de Développement Régional (FEDER) et le Conseil Régional des Hauts-De-France entre autres.

References

- [1] T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009.
- [2] C. Ansótegui and J. Gabàs. Solving (weighted) partial maxsat with ILP. In *Integration of AI and OR Techniques in Constraint Programming*, 2013.
- [3] J. Berg, M. Järvisalo, R. Martins, A. Niskanen, and T. Paxian, editors. *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2024.
- [4] A. Biere, M. Heule, and H. van Maaren. *Handbook of Satisfiability: Second Edition*. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021.
- [5] S. Cherif, H. Sattoutah, C. Li, C. Lucet, and L. B. Devendeville. Minimizing working-group conflicts in conference session scheduling through maximum satisfiability (short paper). In P. Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, September 2–6, 2024, Girona, Spain*, volume 307 of *LIPICs*, pages 34:1–34:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [6] E. Demirović and N. Musliu. Maxsat-based large neighborhood search for high school timetabling. *Computers & Operations Research*, 78:172–180, 2017.
- [7] C. M. Li and F. Manyà. Chapter 23. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press, Feb. 2021.
- [8] C.-M. Li, Z. Xu, J. Coll, F. Manyà, D. Habet, and K. He. Combining Clause Learning and Branch and Bound for MaxSAT. *LIPICs, Volume 210, CP 2021*, 210:38:1–38:18, 2021.
- [9] A. Morgado, C. Dodaro, and J. Marques-Silva. Core-guided maxsat with soft cardinality constraints. In B. O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014.
- [10] R. Ostrowski, É. Grégoire, B. Mazure, and L. Sais. Recovering and exploiting structural knowledge from CNF formulas. In P. V. Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9–13, 2002. Proceedings*, volume 2470 of *Lecture Notes in Computer Science*, pages 185–199. Springer, 2002.
- [11] M. Piotrow. UWMaxSat: Efficient Solver for MaxSAT and Pseudo-Boolean Problems. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 132–136, Baltimore, MD, USA, Nov. 2020. IEEE.
- [12] S. Safarpour, H. Mangassarian, A. Veneris, M. H. Liffiton, and K. A. Sakallah. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer Aided Design (FMCAD'07)*, pages 13–19, 2007.
- [13] M. W. P. Savelsbergh. Preprocessing and Probing Techniques for Mixed Integer Programming Problems. *ORSA Journal on Computing*, 6(4):445–454, Nov. 1994.
- [14] J. P. Wallner, A. Niskanen, and M. Järvisalo. Complexity results and algorithms for extension enforcement in abstract argumentation. *J. Artif. Int. Res.*, 60(1):1–40, Sept. 2017.
- [15] Z. Zheng, S. Cherif, and R. S. Shibusaki. Optimizing power peaks in simple assembly line balancing through maximum satisfiability. In *36th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2024, Herndon, VA, USA, October 28–30, 2024*, pages 363–370. IEEE, 2024.

Formulations MaxSAT pour la programmation non linéaire en nombres entiers

Zhifei Zheng, Sami Cherif, Rui Sá Shibasaki, Chu-Min Li, Jialu Zhang
Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

{zhifei.zheng;sami.cherif;rui.sa.shibasaki;chu-min.li;jialu.zhang}@u-picardie.fr

Résumé

Cet article présente de nouvelles formulations basées sur la Satisfiabilité Maximum (MaxSAT) pour encoder des problèmes de Programmation Non Linéaire en Nombres Entiers (NLIP) avec des fonctions polynomiales. Nous développons un cadre générique pour les fonctions polynomiales discrètes, basé sur trois techniques d'encodage d'entiers issues de la littérature. Notre approche traite tout d'abord chaque terme polynomial comme une unité atomique et démontre comment ces derniers peuvent être efficacement encodés grâce à des représentations compactes d'assignations d'entiers. De plus, nous présentons deux méthodes distinctes de décomposition basées sur les degrés des termes polynomiaux. Ces travaux ont donc le potentiel d'étendre l'applicabilité des solveurs MaxSAT à un éventail plus large de problèmes d'optimisation.

Mots-clés

Satisfiabilité Maximum, Programmation Non Linéaire en Nombres Entiers, Polynômes, Encodages

Abstract

This paper introduces novel Maximum Satisfiability (MaxSAT) formulations for encoding Nonlinear Integer Programming (NLIP) problems with polynomial functions. Specifically designed for discrete polynomial functions, we develop a generic framework based on three established integer encoding techniques from the literature. Our approach treats each polynomial term as an atomic unit and demonstrates how these can be efficiently encoded through compact representations of integer assignments. Additionally, we present two distinct decomposition methods based on polynomial term degrees. This preliminary work lays a foundation for future research and has the potential to extend the applicability of modern MaxSAT solvers to a wider range of optimization problems.

Keywords

Maximum Satisfiability, Nonlinear Integer Programming, Polynomials, Encodings

1 Introduction

La satisfiabilité propositionnelle (SAT) et la satisfiabilité maximum (MaxSAT) sont deux formalismes puissants dotés d'une grande expressivité en logique propositionnelle.

Étant donné une formule propositionnelle sous forme d'un ensemble de clauses, le problème SAT consiste à déterminer si cette formule est satisfiable [7]. À son tour, MaxSAT consiste à maximiser le nombre de clauses satisfaites dans la formule donnée. Ces dernières années, MaxSAT a connu un essor important avec des applications dans de nombreux domaines, notamment la vérification matérielle et logicielle [18, 33], la planification et l'ordonnement [10, 35] et la bio-informatique [16, 17]. Ces applications illustrent la manière dont MaxSAT parvient à réduire l'écart entre les variables booléennes, éléments atomiques fondamentaux de la logique propositionnelle, et les variables entières, davantage représentatives des problématiques issues du monde réel. Par ailleurs, l'introduction de contraintes pseudo-booléennes confère à MaxSAT la capacité de traiter des problèmes de programmation linéaire en nombres entiers. Néanmoins, le potentiel de MaxSAT dans un cadre non linéaire reste peu exploré dans la littérature. Cet article propose donc d'étudier la formulation de problèmes de programmation non linéaire en nombres entiers, exprimés à l'aide de fonctions polynomiales, à travers le prisme de la satisfiabilité maximum.

La programmation non linéaire en nombres entiers (Nonlinear Integer Programming, NLIP) constitue un problème fondamental, avec de nombreuses applications dans des domaines variés [21, 13, 29], puisque de nombreuses situations réelles s'expriment naturellement sur des domaines discrets au sein de structures polynomiales complexes. Cependant, NLIP demeure un problème notoirement difficile, tant sur le plan théorique que pratique. Cette difficulté provient principalement de la combinaison des contraintes d'intégralité et de la non-linéarité des expressions polynomiales. En effet, il a été démontré dans [27] que : (i) l'optimisation entière d'un polynôme quadratique dans une région polygonale convexe est un problème NP-difficile ; (ii) lorsque les variables ne sont pas bornées, même avec une fonction objectif linéaire et des contraintes polynomiales, le problème devient incalculable par une fonction récursive dès lors qu'il implique jusqu'à 10 variables entières. En outre, il a également été établi dans [22] que cette incalculabilité persiste pour des fonctions objectif linéaires soumises à des contraintes quadratiques, dès lors que les variables ne sont pas bornées. Ainsi, nous restreignons notre étude aux cas où les variables entières sont bornées.

Dans cet article, nous proposons des formulations Max-

SAT pour la programmation non linéaire en nombres entiers, fondées sur des représentations entières de la littérature. Tout d'abord, nous traitons chaque terme polynomial comme une unité atomique indépendante, pour laquelle nous élaborons des représentations booléennes. En particulier, nous développons trois formulations différentes basées sur les encodages one-hot, unaire et binaire. De plus, nous examinons les décompositions possibles de l'ordre des termes polynomiaux pour établir des formulations plus compactes. Les formulations proposées dans cet article mettent en évidence le potentiel de MaxSAT pour aborder une classe plus large de problèmes d'optimisation au-delà de leur domaine d'application traditionnel.

Le reste de cet article est organisé comme suit. La section 2 présente les définitions et notations fondamentales de MaxSAT et NLIP; et passe en revue des travaux connexes de la littérature. La section 3 considère chaque terme polynomial comme une unité atomique, utilisant les trois représentations différentes pour encoder un NLIP donné. La section 4 se focalise sur la décomposition du degré des termes polynomiaux donnés, réduisant ainsi la complexité des encodages. La section 5 présente une étude de cas sur les polynômes d'ordre supérieur comme exemple illustratif. Enfin, nous concluons et discutons des travaux futurs dans la section 6.

2 Préliminaires

2.1 Satisfiabilité maximum

La satisfiabilité maximum (MaxSAT) est l'extension de la satisfiabilité (SAT) en problème d'optimisation. Dans cet article, nous utilisons sa version générique, à savoir MaxSAT partiel pondéré [24, 4]. Ce problème prend en entrée une formule bipartite pondérée $\phi = H \cup S$ contenant des variables booléennes dans X . H est l'ensemble des clauses dures qui doivent être satisfaites, comme dans SAT. S est l'ensemble des clauses souples, constituées de clauses pondérées (C, W_C) , où W_C est un poids entier positif associé à la clause C . L'objectif de MaxSAT partiel pondéré est de trouver une affectation des variables α qui maximise (resp. minimise) la somme des poids des clauses souples satisfaites (resp. falsifiées) tout en satisfaisant toutes les clauses dures. Formellement, soit $cost_\alpha(\phi)$ la somme des poids des clauses souples dans ϕ falsifiées par l'affectation α , le problème MaxSAT partiel pondéré cherche à obtenir $optimum(\phi) = \min_\alpha cost_\alpha(\phi)$.

Dans la suite, nous utiliserons simplement MaxSAT pour désigner la version partielle pondérée définie ci-dessus. Il convient de noter que dans MaxSAT, les poids sont considérés comme des entiers positifs. Cependant, dans cet article, nous permettons que les poids soient négatifs, car ils peuvent toujours être normalisés en poids positifs. Enfin, nous mentionnons que la littérature introduit les contraintes Pseudo-Booléennes (PB) [32], qui peuvent être formalisées comme suit :

$$\sum_j a_j \cdot l_j \triangleright b$$

tel que $a_j \in \mathbb{N}, b \in \mathbb{N}, l_j$ est un littéral et $\triangleright \in \{=, \geq, \leq\}$.

2.2 Programmation non linéaire en nombres entiers

Dans cette section, nous définissons formellement les composants de la Programmation Non Linéaire en Nombres Entiers (NLIP) en se focalisant sur les fonctions polynomiales et nous introduisons les terminologies pertinentes utilisées dans l'ensemble de cet article. Comme mentionné dans l'introduction, notre étude se limite aux variables entières bornées inférieurement par 0 et supérieurement par une constante fixée ub . Le domaine des variables entières est donc donné par $V = \{0, 1, \dots, ub\}$. De plus, nous restreignons les coefficients à l'ensemble des entiers $\mathbb{Z}$, car les applications réelles nécessitant une précision infinie sont extrêmement rares.

Définition 1 (Terme polynomial). *Un terme $T(\mathbf{X})$ sur les variables $\mathbf{X} = (X_1, \dots, X_m)$ est appelé terme polynomial s'il a la forme suivante :*

$$T(\mathbf{X}) = c \cdot \prod_{q=q_1}^{q_k} X_q$$

avec $c \in \mathbb{Z}, k \in \mathbb{N}$ et $q_1, \dots, q_k \in \{1, \dots, m\}$.

À noter que, dans la définition ci-dessus, les indices k ne sont pas nécessairement distincts. Nous appelons un terme contenant une seule variable (par exemple X_1^2) un *terme pur*, et un terme contenant plusieurs variables différentes (par exemple $X_1 X_2$) un *terme croisé*. Nous appelons k l'ordre de $T(\mathbf{X})$. En particulier, lorsque $k \geq 2$, nous qualifions ce terme de *terme de grand ordre*. Lorsque le contexte est clair, nous abrégeons $T(\mathbf{X})$ en T . Sans perte de généralité, pour une formulation d'un terme polynomial à m variables et d'ordre k , nous utiliserons également une représentation équivalente $T(\mathbf{X}) = c \cdot \prod_{q=1}^m X_q^{k_q}$, où $\sum_{q=1}^m k_q = k$. En introduisant la notion de termes polynomiaux, nous pouvons maintenant formaliser les fonctions polynomiales (non linéaires) avant de présenter NLIP.

Définition 2 (Fonction polynomiale). *Une fonction $P(\mathbf{X})$ est dite polynomiale si elle peut s'écrire comme la somme d'une série de n termes polynomiaux :*

$$P(\mathbf{X}) = \sum_{p=1}^n T_p(\mathbf{X})$$

où $n \in \mathbb{N}$ et $T_1(\mathbf{X}), \dots, T_n(\mathbf{X})$ sont des termes polynomiaux.

Étant donné $P(\mathbf{X})$, s'il existe p tel que $T_p(\mathbf{X})$ est un terme de grand ordre, alors $P(\mathbf{X})$ est appelée une *fonction polynomiale non linéaire*.

Définition 3 (NLIP). *Un modèle de programmation non linéaire en nombres entiers (NLIP), comportant m variables entières et $\mathcal{M}$ contraintes, peut être formalisé de la manière suivante [25, 20] :*

$$\begin{aligned} \min \quad & f(\mathbf{X}) & \text{(NLIP)} \\ \text{s.t.} \quad & g_i(\mathbf{X}) \leq b_i & i = 1, \dots, \mathcal{M} \\ & \mathbf{X} \in \mathbb{X}, \mathbb{X} \subset \{0, \dots, ub\}^m \end{aligned}$$

où au moins l'une des fonctions f ou g_i est non linéaire ; $\mathbf{X}$ est un vecteur dont chacun des m composants est une variable entière positive ou nulle, bornée supérieurement par ub .

Dans le cadre de cette étude, nous restreignons toutes les fonctions f et g_i à des fonctions polynomiales.

Exemple 1. Ci-dessous, on donne un exemple de problème NLIP à deux variables :

$$\begin{aligned} \min \quad & x_1^2 + x_1x_2 - 2x_2 \\ \text{s.t.} \quad & x_1^2 - x_2 \leq 3 \\ & x_1 + x_2 \leq 5 \quad x_1, x_2 \in \{0, 1, 2, 3, 4\} \end{aligned}$$

2.3 Travaux connexes

Nous avons déjà présenté la définition de la programmation non linéaire en nombres entiers (NLIP) [25, 20]. Lorsque les contraintes entières imposées aux variables sont restreintes à des contraintes booléennes, c'est-à-dire lorsque toutes les variables ne peuvent prendre que les valeurs dans $\{0, 1\}$, le problème NLIP se transforme alors en un problème de programmation non linéaire en 0-1, également connu sous le nom d'optimisation pseudo-booléenne (PBO) [5, 9]. Selon la nature des contraintes, PBO peut être affiné en plusieurs sous-domaines. Par exemple, on distingue le PBO quadratique, où toutes les contraintes sont de degré deux. Le problème MaxSAT pondéré partiel [24, 4] traite également de l'optimisation sur des variables booléennes, mais dans le cadre de la logique propositionnelle, où les contraintes sont exprimées sous forme de clauses dures et la fonction objectif est représentée par des clauses souples.

Beaucoup d'efforts ont été consacrés dans la littérature pour combler le fossé entre les différents types d'optimisation mathématique. Par exemple, [19] présente une méthode simple pour transformer tout problème de programmation entière en un problème de programmation entière 0-1, en remplaçant chaque variable entière par une série de variables booléennes représentant collectivement l'entier d'origine en format binaire. Plusieurs travaux [31, 12, 3, 30] introduisent également des réductions de PBO non linéaire vers des polynômes multilinéaires, ou encore vers PBO quadratique et linéaire. Enfin, plusieurs méthodes existent pour transformer les contraintes PB en clauses [14, 28, 1], ce qui fait le lien entre PBO et MaxSAT. Nous évoquons ici également les travaux connexes de [2], qui explorent la formulation des problèmes linéaires à domaines finis sous forme de problèmes SAT équivalents, en proposant six méthodes de mappage différentes. Cette étude peut également être vue comme une réflexion sur l'encodage des variables entières à l'aide de variables booléennes. Il est également important de noter que, comme le souligne [32], la programmation pseudo-booléenne est un domaine qui n'a pas été aussi développé que celui de la satisfiabilité, ce qui explique pourquoi nous développons notre travail à travers le prisme de MaxSAT.

Dans cet article, nous présentons donc une transformation de bout en bout de NLIP vers MaxSAT. Notre travail

s'appuie sur des travaux antérieurs portant sur les représentations booléennes des entiers, les contraintes pseudo-boléennes ainsi que les techniques de décomposition, afin d'introduire un schéma de formulation systématique de NLIP à l'aide de MaxSAT.

2.4 Représentation des entiers avec MaxSAT

Dans cette section, nous présentons différents encodages des entiers à l'aide de variables booléennes. Plus précisément, étant donné une variable entière naturelle bornée $X \in \mathbb{N}$ telle que $X \leq ub$, où ub est une borne supérieure constante, une série de variables booléennes x_i est utilisée pour représenter X , selon l'une des trois approches couramment employées dans la littérature : la représentation one-hot [15, 23], la représentation unaire [6, 8] et la représentation binaire [14, 36]. Il convient de noter que les représentations one-hot, unaire et binaire correspondent respectivement aux encodages standard (Standard mapping), régulier (Full regular mapping), et logarithmique (Full logarithmic mapping), tels que définis dans [2].

Étant donné un entier naturel $X \leq ub$, les représentations one-hot et unaire utilisent toutes les deux $ub + 1$ variables booléennes pour représenter X . Dans une représentation one-hot, seul le bit dont l'indice correspond à la valeur de X est assigné à 1, tandis que tous les autres sont assignés à 0. En revanche, dans la représentation unaire, tous les bits dont l'indice est inférieur ou égal à la valeur de X sont assignés à 1, les autres étant assignés à 0. Il est important de noter que, puisque $X \in \mathbb{N}$ et que 0 constitue une borne inférieure implicite, la variable x_0 est toujours fixée à 1. En ce qui concerne la représentation binaire, elle utilise $\lfloor \log_2 ub \rfloor + 1$ variables booléennes pour coder X . Les bits correspondent à ceux de l'écriture binaire (c'est-à-dire en base 2) de la valeur de X . Ainsi, les bits sont assignés à 1 s'ils correspondent à des positions actives dans la représentation binaire de la valeur attribuée à X , et à 0 dans le cas contraire.

Exemple 2. Supposons que la borne supérieure de la variable entière soit $ub = 6$, et que la valeur obtenue soit $X = 5$. Dans les représentations one-hot et unaire, il faut $ub + 1 = 7$ variables booléennes pour encoder cette valeur. On obtient alors l'affectation suivante dans la représentation one-hot : $x_0 = 0, x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 1, x_6 = 0$; et l'affectation suivante dans la représentation unaire : $x_0 = 1, x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 1, x_6 = 0$. En revanche, la représentation binaire ne nécessite que $\lfloor \log_2 6 \rfloor + 1 = 3$ variables. Étant donné que $(5)_{10} = (101)_2$, l'affectation correspondante est : $x_2 = 1, x_1 = 0, x_0 = 1$.

Par la suite, lors de la représentation de la variable entière $X_q \in \mathbf{X}$ ($q \in \{1, 2, \dots, m\}$), nous désignons les variables booléennes associées par $x_r^{(q)}$, où r est l'identifiant de la variable. Nous récapitulons l'ensemble des notations et des identifiants utilisés dans le Tableau 1. Sauf indication contraire, $\log$ désignera le logarithme en base 2. En se basant sur les représentations entières introduites précédemment, nous abordons désormais la formulation d'un

TABLE 1 – Notations

Notation	Description
<i>Ensembles</i>	
$\mathbb{N}$	Ensemble des entiers naturels : $\{0, 1, 2, \dots\}$
$\mathbb{Z}$	Ensemble des entiers relatifs : $\{\dots, -2, -1, 0, 1, 2, \dots\}$
<i>Variables et fonctions</i>	
X	Une variable entière
$\mathbf{X}$	Un vecteur de variables entières
$T(\mathbf{X})$	Un terme polynomial sur $\mathbf{X}$ (abrégé en T)
$P(\mathbf{X})$	Une fonction polynomiale non linéaire sur $\mathbf{X}$
x	Une variable booléenne correspondant à X
<i>Indices</i>	
$p, p_1, p_2, \dots$	Indice des termes dans une fonction
$q, q_1, q_2, \dots$	Indices des variables entières dans un terme
$r, r_1, r_2, \dots$	Indices des variables booléennes correspondant à un entier

problème général d'optimisation sous contraintes par une reformulation en MaxSAT.

Un modèle MaxSAT partiel pondéré se compose de clauses souples pondérées et de clauses dures, ce qui correspond dans le cadre de NLIP respectivement à la fonction objectif et aux contraintes : les premières décrivent l'objectif à optimiser, tandis que les secondes définissent l'espace de solution. Plus précisément, si toutes les clauses souples sont des clauses unitaires contenant un seul littéral (ce qui peut toujours être obtenu en remplaçant toute clause non unitaire par un littéral, et en ajoutant des contraintes dures assurant leur cohérence), chacune de ces clauses peut être vue comme un terme de la fonction objectif, avec son poids représentant le coefficient associé. L'exemple 3 illustre une transformation simple d'un modèle de programmation linéaire en un modèle MaxSAT.

Exemple 3. *Le programme linéaire booléen suivant :*

$$\begin{aligned} \max \quad & 2x + 3y \\ \text{s.c} \quad & x + 2y \leq 10 \\ & x, y \in \{0, 1\} \end{aligned}$$

peut être représenté avec le modèle MaxSAT suivant :

$$\begin{aligned} \text{Clauses souples :} \quad & (x, 2), (y, 3) \\ \text{Clauses dures :} \quad & x + 2y \leq 10 \text{ (contrainte PB)} \end{aligned}$$

Ainsi, étant donné une instance de NLIP avec des variables dans $\mathbf{X}$, l'objectif de la procédure de formulation est de transformer la fonction objectif et les contraintes en clauses. Essentiellement, ces tâches peuvent être réduites à la représentation de chaque terme polynomial $T(\mathbf{X})$. Dans la suite, nous abordons cette question sous du point de vue des fonctions objectif, bien qu'une procédure identique puisse être appliquée à la transformation des contraintes.

3 Formulations atomiques des termes non linéaires

Dans cette section, nous nous focalisons sur les termes polynomiaux, considérés comme des unités atomiques. Nous les encodons à l'aide des trois représentations : one-hot, unaire et binaire.

3.1 Formulation one-hot

La représentation one-hot constitue la formulation la plus directe. Nous commençons par le cas le plus simple, à savoir la représentation d'un terme pur sous forme one-hot :

$$T_{\text{pure}} = c \cdot X^k = \sum_{r=0}^{ub} c \cdot r^k \cdot x_r$$

Pour transformer une terme croisé avec deux variables, on peut également procéder comme suit :

$$T_{bi\text{-}cross} = c \cdot X_{q_1}^{k_1} X_{q_2}^{k_2} = \sum_{r_1=0}^{ub} \sum_{r_2=0}^{ub} c \cdot r_1^{k_1} r_2^{k_2} x_{r_1}^{(q_1)} x_{r_2}^{(q_2)}$$

On remarque ici que cette équation contient des termes booléens non linéaires de la forme $x_{r_1}^{(q_1)} x_{r_2}^{(q_2)}$, ce qui constitue une difficulté systématique lorsqu'on traite un terme croisé. Toutefois, il est possible de linéariser ces produits à l'aide d'une nouvelle série de variables booléennes auxiliaires. Prenons l'exemple de $T_{bi\text{-}cross}$: on peut introduire des variables auxiliaires $z_{r_1, r_2} \forall (r_1, r_2) \in V^2$, où $V = \{0, \dots, ub\}$, et il suffit de rajouter les contraintes auxiliaires suivantes :

$$z_{r_1, r_2} \leftrightarrow (x_{r_1}^{(q_1)} \wedge x_{r_2}^{(q_2)}) \quad \forall (r_1, r_2) \in V^2$$

En généralisant la procédure ci-dessus, on peut obtenir la formulation suivante pour tout terme polynomial T comportant m variables distinctes et d'ordre k :

$$\begin{aligned} T &= c \cdot \prod_{q=1}^m X_q^{k_q} \quad (\text{such that } \sum_{q=1}^m k_q = k) \\ &= \sum_{r_1=0}^{ub} \sum_{r_2=0}^{ub} \dots \sum_{r_m=0}^{ub} c \cdot \left(\prod_{q=1}^m r_q^{k_q} \right) \cdot z_{r_1, r_2, \dots, r_m} \quad (\text{OH-1}) \end{aligned}$$

$$z_{r_1, r_2, \dots, r_m} \leftrightarrow \bigwedge_{q=1}^m x_{r_q}^{(q)} \quad \forall (r_1, r_2, \dots, r_m) \in V^m \quad (\text{OH-2})$$

$$\sum_{r \in V} x_r^{(q)} = 1 \quad \forall q \in \{q_1, \dots, q_m\} \quad (\text{OH-3})$$

Les équations OH-1 à OH-3 constituent ainsi la représentation one-hot pour tout terme polynomial. Lorsque T appartient à la fonction objectif donnée, on considère l'équation OH-1 comme les clauses souples suivantes :

$$(z_{r_1, r_2, \dots, r_m}, c \cdot \left(\prod_{q=1}^m r_q^{k_q} \right)) \quad \forall (r_1, r_2, \dots, r_m) \in V^m$$

tandis que OH-2 et OH-3 sont considérées comme contraintes dures.

Exemple 4. *Supposons $T = X_1^2 X_2$ et $ub = 3$. Nous utilisons des variables booléennes $x_r^{(1)}$ pour représenter X_1 et $x_r^{(2)}$ pour représenter X_2 en encodage one-hot. Nous introduisons d'abord les variables booléennes auxiliaires*

z_{r_1, r_2} pour tout $(r_1, r_2) \in V^2 = \{0, 1, 2, 3\}^2$ et ajoutons les contraintes suivantes :

$$\sum_{r_1 \in V} x_{r_1}^{(1)} = 1 \quad \text{et} \quad \sum_{r_2 \in V} x_{r_2}^{(2)} = 1$$

$$z_{r_1, r_2} \leftrightarrow (x_{r_1}^{(1)} \wedge x_{r_2}^{(2)}) \quad \forall (r_1, r_2) \in \{0, 1, 2, 3\}^2$$

Nous pouvons alors représenter T comme suit :

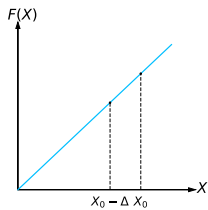
$$\begin{aligned} T = & (0 \times 0) \cdot z_{0,0} + (1 \times 0) \cdot z_{1,0} + (4 \times 0) \cdot z_{2,0} + (9 \times 0) \cdot z_{3,0} \\ & + (0 \times 1) \cdot z_{0,1} + (1 \times 1) \cdot z_{1,1} + (4 \times 1) \cdot z_{2,1} + (9 \times 1) \cdot z_{3,1} \\ & + (0 \times 2) \cdot z_{0,2} + (1 \times 2) \cdot z_{1,2} + (4 \times 2) \cdot z_{2,2} + (9 \times 2) \cdot z_{3,2} \\ & + (0 \times 3) \cdot z_{0,3} + (1 \times 3) \cdot z_{1,3} + (4 \times 3) \cdot z_{2,3} + (9 \times 3) \cdot z_{3,3} \end{aligned}$$

L'avantage de la représentation one-hot réside dans sa simplicité. Cependant, pour encoder un terme comportant m variables et d'ordre k , elle nécessite un nombre de variables auxiliaires de l'ordre de $O(ub^m)$, ainsi que $O(ub^m)$ clauses pour l'encodage des contraintes (et jusqu'à $O(ub^k)$ dans le pire des cas où $k = m$).

3.2 Formulation unaire

Dans cette section, on présente la formulation unaire, fondée sur la variation incrémentale de la valeur d'un terme lorsque les valeurs de ses variables changent. Cela correspond essentiellement à la notion de *différence* définie dans la suite. La théorie des différences remonte à l'application des méthodes de différences finies en analyse discrète [11, 34]. Les différences sont aux fonctions discrètes ce que les dérivées sont aux fonctions continues. De même, les sommes préfixées, définies ci-dessous, sont aux fonctions discrètes ce que les intégrales sont aux fonctions continues. L'analogie entre ces concepts est illustrée dans les figures 1 et 2, qui seront étayés plus en détail dans la Proposition 3.

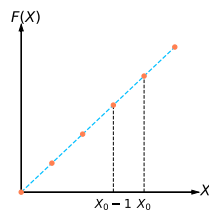
Les différences offrent un cadre naturel pour décomposer les fonctions discrètes en contributions incrémentales, ce qui est particulièrement utile pour représenter les termes polynomiaux dans les formulations MaxSAT. En comprenant comment les valeurs des fonctions changent lorsque les variables augmentent d'une unité, nous pouvons encoder efficacement les variables entières à l'aide de variables booléennes dans une représentation unaire. Formellement, la *différence* d'un terme polynomial T est définie comme suit :



$$F'(X) = \lim_{\Delta \rightarrow 0} \frac{F(X) - F(X - \Delta)}{\Delta}$$

$$\int_0^{X_0} F'(X) dX = F(X_0) - F(0)$$

FIGURE 1 – Fonction continue



$$\tilde{F}(X) = F(X) - F(X - 1)$$

$$\sum_{X=1}^{X_0} \tilde{F}(X) = F(X_0) - F(0)$$

FIGURE 2 – Fonction discrète

Définition 4 (Différence d'un terme). Pour un terme polynomial $T(\mathbf{X}) = c \cdot \prod_{q=1}^m X_q^{k_q}$, sa différence $\tilde{T}$ au point $\mathbf{X} = (r_1, \dots, r_m)$ est définie par :

$$\tilde{T}(r_1, \dots, r_m) = \begin{cases} 0 & \text{si } \exists q \in \{1, \dots, m\} \text{ tel que } r_q = 0; \\ \sum_{S \subseteq \{1, \dots, m\}} (-1)^{|S|} T(r'_1, \dots, r'_m) & \text{sinon} \end{cases}$$

où $r'_q = r_q - 1$ si $q \in S$ et $r'_q = r_q$ si $q \in \{1, \dots, m\} \setminus S$.

En se basant sur la définition de la différence et sur la structure des termes polynomiaux, la Proposition 1 décrit l'échelle de $\tilde{T}$ et la Proposition 2 établit une forme inductive de la fonction de différence pour les termes polynomiaux, en fonction de leur degré.

Proposition 1. Étant donné $T(\mathbf{X})$ d'ordre k , l'ordre de $\tilde{T}(\mathbf{X})$ est inférieur à k .

Preuve. L'ensemble $\{1, \dots, m\}$ a 2^m sous-ensembles distincts, y compris 2^{m-1} sous-ensembles de cardinalité impaire et 2^{m-1} sous-ensembles de cardinalité paire, par conséquent les coefficients des termes d'ordre le plus élevé s'annulent mutuellement dans la somme.

Proposition 2. Étant donné un terme polynomial $T(\mathbf{X}) = c \cdot \prod_{q=1}^m r_q^{k_q}$ et un point fixe $\mathbf{X} = (r_1, \dots, r_m)$ tel que $X_q \neq 0 \forall q \in \{1, \dots, m\}$, on a :

$$\tilde{T}(X) = (r_m^{k_m} - (r_m - 1)^{k_m}) \cdot \tilde{T}(r_1, \dots, r_{m-1})$$

où $T(r_1, \dots, r_{m-1}) = c \cdot \prod_{q=1}^{m-1} r_q^{k_q}$.

Preuve. On peut démontrer ce résultat en raisonnant séparément sur les cas où $m \in S$ ou $m \notin S$, comme suit :

$$\begin{aligned} \tilde{T}(r_1, \dots, r_m) &= \sum_{S \subseteq \{1, \dots, m\}} (-1)^{|S|} T(r'_1, \dots, r'_m) \\ &= \sum_{S \subseteq \{1, \dots, m-1\}} (-1)^{|S|} T(r'_1, \dots, r'_{m-1}) \cdot r_m^{k_m} \\ &\quad + \sum_{S \subseteq \{1, \dots, m-1\}} (-1)^{|S|+1} T(r'_1, \dots, r'_{m-1}) \cdot (r_m - 1)^{k_m} \\ &= \tilde{T}(r_1, \dots, r_{m-1}) \cdot r_m^{k_m} - \tilde{T}(r_1, \dots, r_{m-1}) \cdot (r_m - 1)^{k_m} \\ &= (r_m^{k_m} - (r_m - 1)^{k_m}) \cdot \tilde{T}(r_1, \dots, r_{m-1}) \end{aligned}$$

Cette formule récursive nous permet de calculer la fonction de différence pour des termes de grand ordre, et sera également utile dans les discussions ultérieures. Ayant défini la notion de différence, nous introduisons à présent celle de *somme cumulative*, reflétant l'analogie entre les dérivées et les intégrales dans le calcul différentiel continu. Cette construction sera utilisée pour reconstruire les termes polynomiaux à partir de leurs formes différentielles dans le cadre de l'encodage unaire.

Définition 5 (somme cumulative des différences d'un terme). La somme cumulative des différences d'un terme polynomial $T(\mathbf{X})$ en $\mathbf{X} = (X_1, \dots, X_m)$ est définie par :

$$PreSum_T(\mathbf{X}) = \sum_{r_1=1}^{X_1} \sum_{r_2=1}^{X_2} \dots \sum_{r_m=1}^{X_m} \tilde{T}(r_1, \dots, r_m)$$

La proposition suivante révèle que la relation entre la somme cumulative et la différence est analogue à la relation entre l'intégrale et la dérivée.

Proposition 3. Pour tout terme polynomial T et tout $\mathbf{X} = (X_1, \dots, X_m)$, on a $PreSum_T(\mathbf{X}) = T(\mathbf{X})$.

Preuve. Cette proposition peut être démontrée par récurrence sur la dimension m de $\mathbf{X}$. Lorsque $m = 1$, on a $\tilde{T}(X) = T(X) - T(X - 1)$. Ainsi

$$PreSum_T(X) = \sum_{r=1}^X (T(r) - T(r-1)) = T(X) - T(0) = T(X),$$

ce qui montre que la proposition est vérifiée. Supposons maintenant que la propriété soit vraie pour m , et montrons qu'elle l'est aussi pour $m + 1$. Étant donné $\mathbf{X} = (X_1, \dots, X_{m+1})$, on a :

$$\begin{aligned} PreSum_T(\mathbf{X}) &= \sum_{r_1=1}^{X_1} \dots \sum_{r_m=1}^{X_m} \left(\sum_{r_{m+1}=1}^{X_{m+1}} \tilde{T}(r_1, \dots, r_{m+1}) \right) \\ &= \sum_{r_1=1}^{X_1} \dots \sum_{r_m=1}^{X_m} \left(\sum_{r_{m+1}=1}^{X_{m+1}} (r_{m+1}^{k_{m+1}} - (r_{m+1}-1)^{k_{m+1}}) \cdot \tilde{T}(r_1, \dots, r_m) \right) \\ &= \sum_{r_1=1}^{X_1} \dots \sum_{r_m=1}^{X_m} \left(\tilde{T}(r_1, \dots, r_m) \cdot \sum_{r_{m+1}=1}^{X_{m+1}} (r_{m+1}^{k_{m+1}} - (r_{m+1}-1)^{k_{m+1}}) \right) \\ &= \left(\sum_{r_1=1}^{X_1} \dots \sum_{r_m=1}^{X_m} \tilde{T}(r_1, \dots, r_m) \right) \cdot X_{m+1}^{k_{m+1}} \\ &= T(r_1, \dots, r_m) \cdot X_{m+1}^{k_{m+1}} = T(r_1, \dots, r_{m+1}) = T(\mathbf{X}) \end{aligned}$$

Sur la base des concepts de *différence* et de *somme cumulative*, nous pouvons développer la formulation MaxSAT fondée sur l'encodage unaire pour les termes polynomiaux. Commençons par le cas simple d'un terme pur contenant une seule variable entière X :

$$T_{\text{pure}}(X) = c \cdot X^k$$

Nous avons alors :

$$\tilde{T}_{\text{pure}}(X = r) = T_{\text{pure}}(r) - T_{\text{pure}}(r-1) = c \cdot (r^k - (r-1)^k)$$

D'où, pour un X quelconque :

$$T_{\text{pure}} = \sum_{r=1}^{ub} \tilde{T}(r) \cdot x_r = \sum_{r=1}^{ub} c \cdot (r^k - (r-1)^k) \cdot x_r$$

Un exemple correspondant est donné ci-dessous.

Exemple 5. Soit $T = X^2$ un terme contenant une variable entière X bornée par $ub = 4$. On suppose que l'on utilise des variables booléennes x_r pour représenter X sous encodage unaire. Ainsi, on obtient $T(X) = 1x_1 + 3x_2 + 5x_3 + 7x_4$ comme indiqué dans Table 2.

X	0	1	2	3	4
$T = X^2$	0	1	4	9	16
$\tilde{T}$	-	1	3	5	7

TABLE 2 – Différence unidimensionnelle

La représentation unaire pour les termes purs repose sur la différence unidimensionnelle. Lorsqu'un terme polynomial contient deux variables différentes, on utilise alors la différence bidimensionnelle, comme illustré dans l'Exemple suivant.

Exemple 6. Supposons que $T = X_1^2 X_2$ et que la borne supérieure soit $ub = 3$. Nous utilisons les variables booléennes $x_r^{(1)}$ pour représenter X_1 et $x_r^{(2)}$ pour représenter X_2 selon la représentation unaire. La valeur originale de T ainsi que les valeurs de la différence bidimensionnelle sont données respectivement dans les Tables 3 et 4.

X_1	X_2			
	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	4	8	12
3	0	9	18	27

X_1	X_2			
	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0	3	3	3
3	0	5	5	5

TABLE 3 – Valeurs du terme TABLE 4 – Différence bidimensionnelle

Étant donné une paire de valeurs (X_1, X_2) , la somme cumulative bidimensionnelle de la coordonnée correspondante dans la Table 4 donne exactement la valeur de T . Par exemple, si $X_1 = 3$ et $X_2 = 2$, alors :

$$T(X_1, X_2) = \sum_{r_1=1}^{X_1} \sum_{r_2=1}^{X_2} \tilde{T}(r_1, r_2) = 1+1+3+3+5+5 = 18$$

On remarque que la représentation one-hot pour ce cas entraînerait un poids maximal des clauses souples allant jusqu'à 27, tandis que dans la représentation unaire, le poids maximal n'est que de 5.

En généralisant la procédure ci-dessus, on peut obtenir le modèle suivant pour tout terme polynomial T d'ordre k comportant m variables :

$$\begin{aligned} T &= c \cdot \prod_{q=1}^m X_q^{k_q} \quad (\text{tel que } \sum_{q=1}^m k_q = k) \\ &= \sum_{r_1=1}^{ub} \sum_{r_2=1}^{ub} \dots \sum_{r_m=1}^{ub} \tilde{T}(r_1, \dots, r_m) z_{r_1, \dots, r_m} \quad (\text{U-1}) \end{aligned}$$

avec

$$z_{r_1, \dots, r_m} \leftrightarrow \bigwedge_{q=1}^m x_{r_q}^{(q)} \quad \forall (r_1, \dots, r_m) \in V^m \quad (\text{U-2})$$

$$x_r^{(q)} \rightarrow x_{r-1}^{(q)} \quad \forall (q, r) \in N \times V \quad (\text{U-3})$$

Les équations U-1 à U-3 constituent la représentation unaire de tout terme polynomial. Lorsque T est un élément de la fonction objectif donnée, on considère U-1 comme des contraintes souples, tandis que U-2 et U-3 sont considérées

comme des contraintes dures. La représentation unaire nécessite $O(ub^m)$ variables auxiliaires z et $O(ub^m)$ clauses pour l'encodage des contraintes (et $O(ub^k)$ dans le pire des cas où $k = m$) pour encoder un terme avec m variables et d'ordre k .

Dans la résolution de MaxSAT pondéré, la variance des poids des clauses souples constitue un facteur important influençant la difficulté du problème. Un travail récent [26] propose une approche innovante qui traite ces défis dans MaxSAT pondéré par un mécanisme de verrouillage et de déverrouillage partiel des poids souples. Ce mécanisme met également en évidence un problème majeur : lorsque les poids souples présentent de grandes disparités, le nombre de schémas de partitionnement possibles des poids augmente considérablement, ce qui complexifie l'espace de recherche.

Le principal avantage apporté par la représentation unaire réside donc dans la réduction de l'échelle des poids des clauses souples. Comme démontré dans la Proposition 1, la représentation unaire garantit un ordre de grandeur du poids souple ($O(ub^{k-1})$) toujours inférieur à celui obtenu par l'encodage one-hot ($O(ub^k)$).

3.3 Formulation binaire

Dans cette sous-section, nous présentons la représentation binaire applicable à tout terme polynomial. Nous explorons progressivement les cas des termes d'ordre un, d'ordre deux, puis des ordres supérieurs. Supposons qu'il existe un terme linéaire $T_{pure} = cX$ que l'on souhaite représenter de manière binaire. On aura ainsi :

$$T_{pure} = c \cdot X = \sum_{r \in \{0, \dots, \lfloor \log ub \rfloor\}} c \cdot 2^r x_r$$

La représentation binaire consiste donc à décomposer tout entier X comme la somme d'une série de puissances de 2, ce qui permet d'encoder un terme linéaire avec $O(\log ub)$ variables. Passons maintenant à un autre exemple où l'ordre est égal à 2. Étant donné $T = X_1 X_2$, la représentation binaire de ce terme peut être formulée comme suit :

$$\begin{aligned} X_1 X_2 &= \left(\sum_{r_1=0}^{\lfloor \log ub \rfloor} 2^{r_1} x_{r_1} \right) \left(\sum_{r_2=0}^{\lfloor \log ub \rfloor} 2^{r_2} x_{r_2} \right) \\ &= \sum_{r_1=0}^{\lfloor \log ub \rfloor} \sum_{r_2=0}^{\lfloor \log ub \rfloor} 2^{r_1+r_2} x_{r_1} x_{r_2} \quad (\text{B-qua-cross}) \end{aligned}$$

L'équation B-qua-cross peut être généralisée pour formuler tout terme polynomial T avec m variables différentes et d'ordre k . Dans cette formulation, nous commençons par reformuler $\prod_{q=1}^m X_q^{k_q}$ en $\prod_{q=q'_1}^{q'_k} X_q$, où l'ensemble $\{q'_1, q'_2, \dots, q'_k\}$ contient chaque indice de variable répété

autant de fois que sa puissance dans le terme original.

$$\begin{aligned} T &= c \cdot \prod_{q=1}^m X_q^{k_q} = c \cdot \prod_{q=q'_1}^{q'_k} X_q \quad (\text{tel que } \sum_1^m k_q = k) \\ &= \sum_{r_1=0}^{\lfloor \log_2 ub \rfloor} \sum_{r_2=0}^{\lfloor \log_2 ub \rfloor} \cdots \sum_{r_k=0}^{\lfloor \log_2 ub \rfloor} c \cdot \left(\prod_{q=q'_1}^{q'_k} 2^{r_q} \right) \cdot z_{r_1, r_2, \dots, r_k} \end{aligned} \quad (\text{B-1})$$

$$z_{r_1, \dots, r_k} \leftrightarrow \bigwedge_{q=q'_1}^{q'_k} x_{r_q}^{(q)} \quad \forall (r_1, \dots, r_k) \in \{0, \dots, \lfloor \log ub \rfloor\}^k \quad (\text{B-2})$$

Les équations B-1 et B-2 constituent la représentation binaire de tout terme polynomial. Lorsque T est un élément de la fonction objectif considérée, l'équation B-1 est prise comme contrainte souple, tandis que l'équation B-2 est considérée comme contrainte dure.

Exemple 7. Supposons que $T = X_1^2 X_2$ et que $ub = 3$. Nous utilisons des variables booléennes $x_r^{(1)}$ pour représenter X_1 et $x_r^{(2)}$ pour représenter X_2 dans l'encodage binaire. Comme $\lfloor \log ub \rfloor + 1 = \lfloor \log 3 \rfloor + 1 = 2$, nous avons besoin de 2 variables booléennes pour chaque variable entière, indexées par 0 et 1. La représentation binaire nécessite de réécrire $T = X_1 X_1 X_2$. Nous introduisons alors les variables booléennes auxiliaires z_{r_1, r_2, r_3} pour tout $(r_1, r_2, r_3) \in \{0, 1\}^3$ et ajoutons les contraintes suivantes :

$$z_{r_1, r_2, r_3} \leftrightarrow (x_{r_1}^{(1)} \wedge x_{r_2}^{(1)} \wedge x_{r_3}^{(2)}) \quad \forall (r_1, r_2, r_3) \in \{0, 1\}^3$$

Nous pouvons alors représenter T comme suit :

$$\begin{aligned} T &= 2^0 \cdot z_{0,0,0} + 2^1 \cdot z_{0,0,1} + 2^1 \cdot z_{0,1,0} + 2^2 \cdot z_{0,1,1} \\ &\quad + 2^1 \cdot z_{1,0,0} + 2^2 \cdot z_{1,0,1} + 2^2 \cdot z_{1,1,0} + 2^3 \cdot z_{1,1,1} \end{aligned}$$

Par exemple, supposons que $X_1 = 3 = 2^1 + 2^0$ et que $X_2 = 2 = 2^1$. Ainsi, $x_1^{(1)} = 1$, $x_0^{(1)} = 1$, $x_1^{(2)} = 1$, $x_0^{(2)} = 0$ et donc on a :

$$\begin{aligned} T(X_1, X_2) &= \sum_{r_1=0}^{\lfloor \log_2 ub \rfloor} \sum_{r_2=0}^{\lfloor \log_2 ub \rfloor} \sum_{r_3=0}^{\lfloor \log_2 ub \rfloor} \left(\prod_{q=1}^3 2^{r_q} \right) \cdot z_{r_1, r_2, r_3} \\ &= 2^1 \cdot z_{0,0,1} + 2^2 \cdot z_{0,1,1} + 2^2 \cdot z_{1,0,1} + 2^3 \cdot z_{1,1,1} \\ &= 2 + 4 + 4 + 8 = 18 \end{aligned}$$

ce qui correspond bien à $3 \cdot 3 \cdot 2 = (1+2) \cdot (1+2) \cdot (0+2)$.

Nous notons ici une différence majeure des formulations one-hot et unaire. La formulation binaire met principalement l'accent sur le degré k , plutôt que sur le nombre de variables distinctes m impliquées dans l'expression, ce qui explique pourquoi elle commence par la reformulation de $\prod_{q=1}^m X_q^{k_q} = \prod_{q=q'_1}^{q'_k} X_q$. L'exemple suivant illustre la nécessité de cette approche.

Exemple 8. Soit $T(X) = X^2$. On note que $T(4) = 4^2 = 16$, mais $T(1+3) = 1^2 + 3^2 = 10 \neq 16$. Ainsi, on peut seulement reformuler ce terme comme suit :

$$X^2 = \left(\sum_{r=0}^{\lfloor \log ub \rfloor} 2^r x_r \right)^2 = \sum_{r=0}^{\lfloor \log ub \rfloor} 2^{2r} x_r + 2 \sum_{0 \leq r_1 < r_2 \leq \lfloor \log ub \rfloor} 2^{r_1+r_2} x_{r_1} x_{r_2} \quad (\text{B-qua-pure})$$

L'Équation B-qua-pure peut être considérée comme un cas particulier de l'Équation B-qua-cross.

Pour encoder un terme avec m variables et un ordre k , la représentation binaire nécessite :

- $O((\log ub)^k)$ variables auxiliaires z
- $O((\log ub)^k)$ clauses pour encoder les contraintes.

Nous remarquons que, dans la représentation binaire, c'est l'ordre k qui affecte la complexité, contrairement aux représentations one-hot et unaire où c'est le nombre de variables distinctes m qui a un impact ; cependant, elle parvient à réduire la base de la complexité exponentielle à $\log ub$ au lieu de ub dans les encodages précédents.

Nous résumons les complexités des trois représentations présentées dans Table 5. Les trois méthodes de représentation offrent différents compromis en termes d'avantages et d'inconvénients. La représentation one-hot offre une implémentation simple, mais souffre d'une complexité élevée à mesure que le nombre de variables augmente. L'avantage clé de la représentation unitaire est la réduction de l'échelle du poids des clauses souples. La représentation binaire offre l'encodage le plus compact avec une mise à l'échelle logarithmique, mais devient plus complexe lorsqu'il s'agit de termes de grand ordre contenant seulement quelques variables différentes. Ces caractéristiques guident le choix des méthodes d'encodage appropriées en fonction de la structure spécifique des termes polynomiaux dans les applications pratiques. Malgré la complexité quantifiée, il est également important que la représentation soit compatible avec l'information sémantique originale du problème formulé.

4 Stratégies de décomposition pour les termes non linéaires

4.1 Au-delà du traitement atomique

Dans la section précédente, nous avons exploré les formulations atomiques basées sur trois représentations dif-

Aspect	One-hot	Unaire	Binaire
Variables Auxiliaires	$O(ub^m)$	$O(ub^m)$	$O((\log ub)^k)$
Contraintes	$O(ub^m)$	$O(ub^m)$	$O((\log ub)^k)$
Poids maximal	$O(ub^k)$	$O(ub^{k-1})$	$O(ub^k)$

TABLE 5 – Analyse de complexité des différentes formulations atomiques, où m est le nombre de variables distinctes dans le terme, $k \geq m$ est l'ordre du terme et ub est la borne supérieure des variables entières.

férentes, consistant à énumérer toutes les affectations possibles des valeurs des variables pour le terme polynomial donné avec m variables et d'ordre k , entraînant des complexités en termes de variables/contraintes de $O(ub^m)$, $O(ub^m)$ et $O((\log ub)^k)$ respectivement. Avec l'augmentation du nombre de variables (l'ordre), la complexité augmente de manière exponentielle et conduit à une explosion combinatoire. Cette explosion est initialement générée par le fait de traiter le terme polynomial comme une instance atomique et d'énumérer toutes les valeurs possibles. Dans cette section, nous essayons de décomposer un terme en plusieurs sous-unités, dont le produit reste égal au terme original. Plus précisément, étant donné un terme polynomial $T(\mathbf{X})$, la décomposition séquentielle présentée dans la section 4.2 calcule itérativement chaque produit cumulatif de $T(\mathbf{X})$, tandis que la décomposition binaire présentée dans la section 4.3 calcule récursivement le produit des unités voisines.

Exemple 9. Considérons un terme polynomial $T = X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8$. Les Figures 3 et 4 illustrent comment la décomposition séquentielle et la décomposition binaire traiteraient ce terme de manière différente. Comme montré dans la Figure 3, la décomposition séquentielle traite les variables de manière séquentielle de gauche à droite, tandis que la décomposition binaire présentée dans la Figure 4 permet un traitement parallèle à chaque niveau. Les deux méthodes nécessitent 7 multiplications.

Les deux méthodes de décomposition laissent deux degrés de liberté : le choix de la représentation des entiers et le plan de la décomposition. En ce qui concerne le premier choix, les trois méthodes de représentation des entiers — one-hot, unitaire et binaire — peuvent être appliquées dans chacune des stratégies de décomposition. Par souci de clarté et de concision, nous concentrons notre discussion sur la tech-

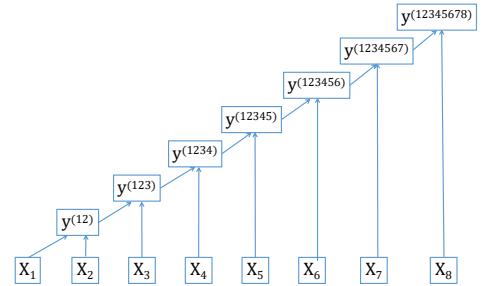


FIGURE 3 – Décomposition séquentielle de l'ordre

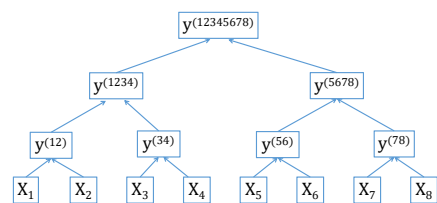


FIGURE 4 – Décomposition binaire de l'ordre

nique de représentation binaire. Le second choix concerne la manière de structurer l'arbre de décomposition lui-même et, plus spécifiquement, l'ordre et le regroupement des variables. Ce volet reste une question ouverte avec un potentiel d'optimisation important. Il est à noter que la représentation binaire offre une flexibilité supplémentaire : lorsque plusieurs termes polynomiaux partagent des sous-groupes de variables communes, ces sous-groupes peuvent être représentés par les mêmes nœuds dans leurs arbres de décomposition respectifs, réduisant ainsi potentiellement la redondance dans l'encodage. L'exemple présenté plus tard dans la section 5 illustrera cette pertinence.

De plus, afin de simplifier la discussion suivante, nous introduisons la notation $Mul(x^{(1)}, x^{(2)})$. Supposons que $x^{(1)}$ et $x^{(2)}$ soient deux séries de variables binaires représentant respectivement les deux entiers X_1 borné par ub_1 et X_2 borné par ub_2 sous la représentation binaire. $Mul(x^{(1)}, x^{(2)})$ définit la formulation binaire du produit de ces deux entiers. Selon l'Équation B-qua-cross, on a :

$$Mul(x^{(1)}, x^{(2)}) = \sum_{r_1=0}^{\lfloor \log ub_1 \rfloor} \sum_{r_2=0}^{\lfloor \log ub_2 \rfloor} 2^{r_1+r_2} z_{r_1 r_2} \quad (\text{Mul-B-1})$$

$$z_{r_1 r_2} \leftrightarrow (x_{r_1}^{(1)} \wedge x_{r_2}^{(2)}) \quad (\text{Mul-B-2})$$

$$\forall (r_1, r_2) \in \{0, \dots, \lfloor \log ub_1 \rfloor\} \times \{0, \dots, \lfloor \log ub_2 \rfloor\}$$

Nous remarquons que chaque appel de Mul utilise implicitement la formulation binaire pour un terme quadratique, et nécessite donc $O(\log ub_1 \cdot \log ub_2)$ variables auxiliaires et clauses.

4.2 Décomposition séquentielle

Dans cette section, nous présentons la décomposition séquentielle, qui traite le terme polynomial de manière séquentielle, calcule chaque produit cumulatif, et introduit une nouvelle série de variables y_r pour chaque opération de multiplication. Formellement, étant donné un terme polynomial $T(\mathbf{X}) = \prod_{q=1}^m X_q^{k_q}$, nous le réécrivons d'abord sous la forme $\prod_{q=q'_1}^{q'_k} X_q$ (tel que $\sum_{q=1}^m k_q = k$), puis nous continuons la procédure comme suit :

$$T(\mathbf{X}) = \prod_{q=1}^m X_q^{k_q} = \prod_{q=q'_1}^{q'_k} X_q \quad (\text{tel que } \sum_{q=1}^m k_q = k)$$

$$\sum_{r=0}^{\lfloor \log(ub^2) \rfloor} 2^r y_r^{(q'_1 q'_2)} = Mul(x^{(q'_1)}, x^{(q'_2)}) \quad (\text{LD-1})$$

$$\sum_{r=0}^{\lfloor \log(ub^k) \rfloor} 2^r y_r^{(q'_1 \dots q'_k)} = Mul(y^{(q'_1 \dots q'_{k-1})}, x^{(q'_k)}) \quad \forall k \in \{3, \dots, k\} \quad (\text{LD-2})$$

$$T(\mathbf{X}) = \sum_{r=0}^{\lfloor \log(ub^k) \rfloor} 2^r y_r^{(q'_1 \dots q'_k)} \quad (\text{LD-3})$$

Les équations LD-1 à LD-3 constituent la représentation de la décomposition séquentielle des termes de grand ordre.

L'équation LD-3 décrit le remplacement du terme de grand ordre apparu par la somme pondérée des variables booléennes représentées en binaire $y^{(q'_1 \dots q'_k)}$. Les équations LD-1 et LD-2 sont utilisées pour maintenir de manière itérative le produit cumulatif du terme donné avec un total de $k - 1$ multiplications, chaque contrainte de celles-ci pouvant être formulée comme une contrainte pseudo-booléenne avec jusqu'à $O(k^2 \cdot \log^2 ub)$ littéraux et un poids maximal inférieur à ub^k . L'échelle du nombre de clauses utilisées est corrélée à la méthode d'encodage PB choisie. Par exemple, si on considère l'encodage PB classique "binary merge" [28], qui parvient à encoder une contrainte pseudo-booléenne avec $O(\mathcal{N} \log^2(\mathcal{N}) \log(\mathcal{W}_{max}))$ clauses, où $\mathcal{N}$ est le nombre de littéraux et $\mathcal{W}_{max}$ est la valeur maximale des poids, cela mène finalement à une complexité globale de $O(k^4 \log^3 ub \cdot \log^2(k^2 \log^2 ub))$ clauses dans la formulation par décomposition séquentielle.

4.3 Décomposition binaire

La décomposition binaire offre une autre perspective pour traiter les termes de grand ordre, où nous divisons récursivement le produit en deux parties de taille similaire jusqu'à ce que chaque partie devienne une seule variable. L'idée principale est de diviser récursivement le produit en deux parties à peu près égales, en calculant le produit de chaque partie séparément, puis en les combinant. Cela forme une structure d'arbre binaire où chaque nœud interne représente le produit de ses deux enfants, comme illustré dans la Figure 4.

Étant donné un terme polynomial $T(\mathbf{X}) = \prod_{q=1}^m X_q^{k_q}$, nous le réécrivons d'abord sous la forme $\prod_{q=q'_1}^{q'_k} X_q$ (tel que $\sum_{q=1}^m k_q = k$), puis nous appliquons la décomposition binaire. Cette procédure commence par obtenir la valeur de $\mathcal{P}$, représentant l'ensemble de toutes les paires d'intervalles adjacents devant être multipliées dans l'arbre binaire. Formellement, $\mathcal{P}$ peut être défini comme suit :

$$\mathcal{P} = \{(\mathcal{I}, \mathcal{J}) \mid \mathcal{I}, \mathcal{J} \subset \{q'_1, q'_2, \dots, q'_k\}, \mathcal{I} \cap \mathcal{J} = \emptyset, \mathcal{I} \cup \mathcal{J} \in \mathcal{T}\}$$

où $\mathcal{T}$ est l'ensemble de tous les nœuds non-feuilles dans l'arbre de décomposition binaire, et pour chaque nœud $N \in \mathcal{T}$, il existe exactement une paire $(\mathcal{I}, \mathcal{J}) \in \mathcal{P}$ telle que $\mathcal{I} \cup \mathcal{J} = N$ et que $\mathcal{I}$ et $\mathcal{J}$ sont des enfants de N dans l'arbre.

$$T(\mathbf{X}) = \prod_{q=q'_1}^{q'_k} X_q = \prod_{q=q'_1}^{q'_{\lceil k/2 \rceil}} X_q \cdot \prod_{q=q'_{\lceil k/2 \rceil+1}}^{q'_k} X_q$$

$$y_r^{(q)} = x_r^{(q)} \quad \forall q \in \{q'_1, \dots, q'_k\}, \quad \forall r \in \{0, \dots, \lfloor \log(ub^k) \rfloor\} \quad (\text{BD-1})$$

$$\sum_{r=0}^{\lfloor \log(ub^{|\mathcal{I}|+|\mathcal{J}|}) \rfloor} 2^r y_r^{(\mathcal{I}, \mathcal{J})} = Mul(y^{(\mathcal{I})}, y^{(\mathcal{J})}) \quad \forall (\mathcal{I}, \mathcal{J}) \in \mathcal{P} \quad (\text{BD-2})$$

$$T(\mathbf{X}) = \sum_{r=0}^{\lfloor \log(ub^k) \rfloor} 2^r y_r^{(q'_1 \dots q'_k)} \quad (\text{BD-3})$$

Exemple 10. Étant donné $k = 8$, l'ensemble $\mathcal{P}$ obtenu inclut les paires d'intervalles suivantes :

- Niveau 1 : $(q'_1, q'_2), (q'_3, q'_4), (q'_5, q'_6), (q'_7, q'_8)$
- Niveau 2 : $(q'_1q'_2, q'_3q'_4), (q'_5q'_6, q'_7q'_8)$
- Niveau 3 : $(q'_1q'_2q'_3q'_4, q'_5q'_6q'_7q'_8)$

L'équation BD-1 initialise les variables auxiliaires $y_r^{(q)}$ avec les variables d'origine $x_r^{(q)}$. L'équation BD-2 combine récursivement les paires de sous-produits en utilisant l'opération Mul défini précédemment, et l'équation BD-3 représente le résultat final. La principale différence entre la décomposition binaire et la décomposition séquentielle réside dans leurs modèles de multiplication. L'échelle du nombre de sommets dans un arbre binaire avec k sommets est $O(k)$, donc il y a $O(k)$ procédures de multiplication dans la procédure de décomposition binaire, ce qui est aligné avec la décomposition séquentielle proposée dans la sous-section précédente. De manière similaire à la décomposition séquentielle, lorsqu'on considère l'encodage PB avec la méthode "Binary Merger", la complexité finale devient $O(k^4 \log^3 ub \cdot \log^2(k^2 \log^2 ub))$. Bien que la complexité générale soit la même, nous notons que la décomposition binaire maintient des distances moyennes plus courtes entre les valeurs entières originales et le produit général, ce qui conduit intuitivement à des lignes d'inférence plus courtes lorsque le modèle est résolu.

5 Étude de cas

Dans cette section, nous étudions la pertinence de la formulation proposée sur la fonction polynomiale de grand ordre $P(\mathbf{X}) = X_1 X_2 X_3 X_4 + X_3 X_4 X_5 X_6 + X_1 X_2 X_5 X_6$, où toutes les variables sont bornées par ub . Limités par l'espace, nous ne présentons ici que la formulation basée sur la décomposition binaire. Dans ce cas spécifique de P , un plan de partition est présenté dans la Figure 5. En conséquence, nous pouvons maintenir les ensembles $\mathcal{P}$ pour les trois termes respectivement, c'est-à-dire : $P_{1234} = \{(1, 2), (3, 4), (12, 34)\}$, $P_{3456} = \{(3, 4), (5, 6), (34, 56)\}$, et $P_{1256} = \{(1, 2), (5, 6), (12, 56)\}$.

Pour chaque élément de $\mathcal{P}$, une nouvelle série de variables auxiliaires est introduite. Cependant, grâce à la structure spécifique et au plan de partitionnement choisi, nous parvenons à utiliser efficacement les variables auxiliaires intermédiaires $y^{(12)}$, $y^{(34)}$ et $y^{(56)}$, chacune servant de com-

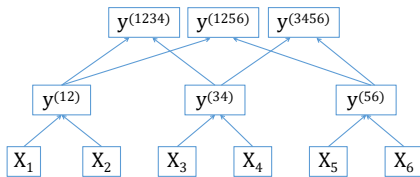


FIGURE 5 – Un plan de décomposition binaire compact pour un polynôme d'ordre supérieur

posant dans deux termes différents. À partir de ce plan de décomposition, nous pouvons construire les termes du polynôme en utilisant l'une des trois représentations suivantes : one-hot, unaire ou binaire, chacune impliquant une échelle différente de variables auxiliaires.

La formulation binaire dans le cadre de la décomposition binaire a été introduite dans la sous-section 4.3. Nous présentons ici simplement la formulation correspondante lorsqu'elle est appliquée au polynôme P .

$$y_r^{(q)} = x_r^{(q)} \quad \forall (q, r) \in \{1, \dots, 6\} \times \{0, \dots, \lfloor \log ub \rfloor\} \quad (\text{BD-B-1})$$

$$\sum_{r=0}^{\lfloor \log(ub^{|I|+|J|}) \rfloor} 2^r y_r^{(\mathcal{I}, \mathcal{J})} = Mul(y^{(\mathcal{I})}, y^{(\mathcal{J})}) \quad (\mathcal{I}, \mathcal{J}) \in \mathcal{P}_{1234} \cup \mathcal{P}_{3456} \cup \mathcal{P}_{1256} \quad (\text{BD-B-2})$$

$$P(X) = \sum_{r=0}^{\lfloor \log(ub^4) \rfloor} 2^r (y_r^{(1234)} + y_r^{(3456)} + y_r^{(1256)}) \quad (\text{BD-B-3})$$

L'équation BD-B-1 représente la base de la structure récursive de la décomposition binaire, tandis que l'équation BD-B-2 définit une étape récursive obtenue par la représentation binaire, reliant deux niveaux de variables auxiliaires. L'équation BD-B-3 décrit la reformulation finale du polynôme.

Nous remarquons que l'opération de multiplication (Mul) joue un rôle essentiel dans la formulation. En raison de contraintes d'espace, nous nous limitons dans la suite à la présentation de la formulation unaire, celle basée sur one-hot pouvant être modélisée de manière analogue. Pour obtenir la formulation correspondante, nous définissons d'abord l'opération de multiplication correspondante avec la formulation unaire comme suit :

$$Mul^{(U)}(x^{(1)}, x^{(2)}) = \sum_{r_1=0}^{ub_1} \sum_{r_2=0}^{ub_2} \tilde{T}(r_1) \cdot \tilde{T}(r_2) z_{r_1 r_2} \quad (\text{Mul-U-1})$$

$$z_{r_1 r_2} \leftrightarrow (x_{r_1}^{(1)} \wedge x_{r_2}^{(2)}) \quad \forall (r_1, r_2) \in \{0, \dots, ub_1\} \times \{0, \dots, ub_2\} \quad (\text{Mul-U-2})$$

Ainsi, nous pouvons établir le plan de formulation suivant pour le polynôme $P(\mathbf{X})$ en utilisant la formulation unaire dans le cadre d'une décomposition binaire :

$$y_r^{(\mathcal{I}, \mathcal{J})} \rightarrow y_{r-1}^{(\mathcal{I}, \mathcal{J})} \quad \forall r \in \{1, \dots, ub^{|\mathcal{I}|+|\mathcal{J}|}\} \quad \forall (\mathcal{I}, \mathcal{J}) \in \mathcal{P}_{1234} \cup \mathcal{P}_{3456} \cup \mathcal{P}_{1256} \quad (\text{BD-U-0})$$

$$y_r^{(q)} = x_r^{(q)} \quad \forall (q, r) \in \{1, \dots, 6\} \times V \quad (\text{BD-U-1})$$

$$\sum_{r=0}^{ub^{|\mathcal{I}|+|\mathcal{J}|}} \tilde{T}(r) \cdot y_r^{(\mathcal{I}, \mathcal{J})} = Mul^{(U)}(y^{(\mathcal{I})}, y^{(\mathcal{J})}) \quad (\mathcal{I}, \mathcal{J}) \in \mathcal{P}_{1234} \cup \mathcal{P}_{3456} \cup \mathcal{P}_{1256} \quad (\text{BD-U-2})$$

$$P(X) = \sum_{r_1=0}^{ub^4} \tilde{T}(r_1) (y_{r_1}^{(1234)} + y_{r_1}^{(3456)} + y_{r_1}^{(1256)}) \quad (\text{BD-U-3})$$

6 Conclusion

Dans cet article, nous avons étudié différentes formulations fondées sur MaxSAT pour les programmes entiers non linéaires comportant des termes polynomiaux. Nous avons d'abord présenté des encodages pour les termes polynomiaux, en proposant trois stratégies différentes, adaptées à divers cas d'usage et avec des complexités différentes. Nous nous sommes ensuite concentrés sur la décomposition des termes selon leur ordre, ce qui permet d'obtenir des formulations à complexité réduite. En combinant ces deux approches, nous proposons ainsi un schéma de formulation générique applicable à toute instance de programmation entière non linéaire incluant des termes polynomiaux.

Comme perspectives, il serait intéressant de continuer à explorer les trois autres méthodes de mappage proposées dans [2], ainsi que d'affiner la stratégie de sélection des méthodes de représentation/décomposition. Sur le plan pratique, après la maturité de la méthode de formulation proposée, une mise en œuvre approfondie des formulations pourrait être appliquée à différents problèmes du monde réel, élargissant ainsi le champ d'application des technologies modernes SAT et MaxSAT. Nous espérons également développer une bibliothèque permettant une utilisation plus facile des formulations et qui sera mise à disposition de la communauté.

Remerciements

Ce travail est partiellement soutenu par le projet ANR-24-CE23-6126 (BforSAT) et la chaire IA ANR-19-CHIA-0013-01 (MASSAL'IA) co-financée par l'agence nationale de recherche et le gestionnaire du réseau de distribution d'électricité Enedis.

Références

- [1] I. Abío, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Bdds for pseudo-boolean constraints - revisited. In K. A. Sakallah and L. Simon, editors, *SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011*, volume 6695 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2011.
- [2] C. Ansótegui and F. Manyà. Mapping problems with finite-domain variables into problems with boolean variables. In *SAT 2004, 10-13 May 2004, Vancouver, BC, Canada, 2004*.
- [3] M. Anthony, E. Boros, Y. Crama, and A. Gruber. Quadratic reformulations of nonlinear binary optimization problems. *Math. Program.*, 162(1-2) :115–144, 2017.
- [4] F. Bacchus, M. Järvisalo, and R. Martins. Maximum satisfiability. In *Handbook of satisfiability*, pages 929–991. IOS Press, 2021.
- [5] E. Balas and J. B. Mazzola. Nonlinear 0-1 programming : I. linearization techniques. *Math. Program.*, 30(1) :1–21, 1984.
- [6] C. Barrett and S. Berezin. A proof-producing boolean search engine. *Program Committee*, page 25, 2003.
- [7] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.
- [8] M. Bofill, J. Coll, M. Garcia, J. Giráldez-Cru, G. Pesant, J. Suy, and M. Villaret. Constraint solving approaches to the business-to-business meeting scheduling problem. *J. Artif. Intell. Res.*, 74 :263–301, 2022.
- [9] E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discret. Appl. Math.*, 123(1-3) :155–225, 2002.
- [10] S. Cherif, H. Sattoutah, C. Li, C. Lucet, and L. B. Devendeville. Minimizing working-group conflicts in conference session scheduling through maximum satisfiability (short paper). In P. Shaw, editor, *CP 2024, September 2-6, 2024, Girona, Spain*, volume 307 of *LIPICs*, pages 34 :1–34 :11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [11] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische annalen*, 100(1) :32–74, 1928.
- [12] Y. Crama and P. L. Hammer. *Boolean Functions - Theory, Algorithms, and Applications*, volume 142 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2011.
- [13] M. A. Duran and I. E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Math. Program.*, 39(3) :337, 1987.
- [14] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *J. Satisf. Boolean Model. Comput.*, 2(1-4) :1–26, 2006.
- [15] A. V. Gelder. Another look at graph coloring via propositional satisfiability. *Discret. Appl. Math.*, 156(2) :230–243, 2008.
- [16] A. Graça, I. Lynce, J. Marques-Silva, and A. L. Oliveira. Efficient and accurate haplotype inference by combining parsimony and pedigree information. In K. Horimoto, M. Nakatsui, and N. Popov, editors, *ANB 2010, Hagenberg, Austria, July 31- August 2, 2010*, volume 6479 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2010.
- [17] J. Guerra and I. Lynce. Reasoning over biological networks using maximum satisfiability. In M. Milano, editor, *CP 2012, Québec City, QC, Canada, October 8-12, 2012*, volume 7514 of *Lecture Notes in Computer Science*, pages 941–956. Springer, 2012.
- [18] A. Gupta, M. K. Ganai, and C. Wang. Sat-based verification methods and applications in hardware verification. In M. Bernardo and A. Cimatti, editors, *SFM 2006, Bertinoro, Italy, May 22-27, 2006, Advanced Lectures*, volume 3965 of *Lecture Notes in Computer Science*, pages 108–143. Springer, 2006.
- [19] P. Hammer Ivănescu and S. Rudeanu. Boolean methods in operations research and related areas. *Oekonometrie und Unternehmensforschung* (, 7, 1968.
- [20] R. Hemmecke, M. Köppe, J. Lee, and R. Weismantel. Nonlinear integer programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 561–618. Springer, 2010.
- [21] M. HM. Portfolio selection : efficient diversification of investments, 1959.
- [22] R. G. Jeroslow. There cannot be any algorithm for integer programming with quadratic constraints. *Oper. Res.*, 21(1) :221–224, 1973.
- [23] W. Klieber and G. Kwon. Efficient cnf encoding for selecting 1 from n objects. In *Proc. International Workshop on Constraints in Formal Verification*, page 14, 2007.
- [24] C. M. Li and F. Manyà. MaxSAT, Hard and Soft Constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 903–927. IOS Press, 2021.
- [25] D. Li, X. Sun, et al. *Nonlinear integer programming*, volume 84. Springer, 2006.
- [26] S. Li, C. Li, J. Coll, D. Habet, and F. Manyà. Improving the lower bound in branch-and-bound algorithms for maxsat. In T. Walsh, J. Shah, and Z. Kolter, editors, *AAAI-25, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 11272–11281. AAAI Press, 2025.
- [27] J. A. D. Loera, R. Hemmecke, M. Köppe, and R. Weismantel. Integer polynomial optimization in fixed dimension. *Math. Oper. Res.*, 31(1) :147–153, 2006.
- [28] N. Manthey, T. Philipp, and P. Steinke. A more compact translation of pseudo-boolean constraints into CNF such that generalized arc consistency is maintained. In C. Lutz and M. Thielscher, editors, *KI 2014 : Advances in Artificial Intelligence, Stuttgart, Germany, September 22-26, 2014*, volume 8736 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2014.
- [29] P. M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem : A survey and recent developments. 1994.
- [30] E. Rodríguez-Heck. Linear and quadratic reformulations of nonlinear optimization problems in binary variables. *4OR*, 17(2) :221–222, 2019.
- [31] I. G. Rosenberg. Reduction of bivalent maximization to the quadratic case. 1975.
- [32] O. Roussel and V. Manquinho. Pseudo-boolean and cardinality constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1087–1129. IOS Press, 2021.
- [33] S. Safarpour, H. Mangassarian, A. G. Veneris, M. H. Liffiton, and K. A. Sakallah. Improved design debugging using maximum satisfiability. In *FMCAD 2007, Austin, Texas, USA, November 11-14, 2007*, pages 13–19. IEEE Computer Society, 2007.
- [34] J. C. Strikwerda. *Finite difference schemes and partial differential equations*. SIAM, 2004.
- [35] P. Surynek. Lazy compilation of variants of multi-robot path planning with satisfiability modulo theory (SMT) approach. In *IROS 2019, Macau, SAR, China, November 3-8, 2019*, pages 3282–3287. IEEE, 2019.
- [36] Z. Zheng, S. Cherif, and R. S. Shibusaki. Minimisation des pics d'énergie dans l'équilibrage des lignes d'assemblage par la satisfiabilité maximum. In *Actes JFPC 2024*, 2024.

**Session 6, mardi 1 juillet, 16h30-17h30 : PPC & Apprentissage
Automatique**

Ensemble de correction minimal basé sur l'adéquation des contraintes en clustering

Aymeric Beauchamp¹, Thi-Bich-Hanh Dao¹, Samir Loudni^{2,3}, Christel Vrain¹

¹Université d'Orléans, INSA CVL, LIFO UR 4022

²IMT Atlantique Bretagne-Pays de la Loire, TASC (LS2N-CNRS)

³Université de Caen-Normandie, GREYC

¹{aymeric.beauchamp, thi-bich-hanh.dao, christel.vrain}@univ-orleans.fr

²samir.loudni@imt-atlantique.fr

Résumé

Le clustering sous contraintes incrémentales voit un expert améliorer pas à pas une partition de départ par ajouts successifs de contraintes. Un conflit peut apparaître quand des contraintes ajoutées entrent en contradiction avec d'autres fournies aux itérations précédentes. Pour résoudre ce problème, nous proposons une approche hybride combinant apprentissage automatique et programmation par contraintes. Notre méthode consiste à entraîner un modèle sur les retours utilisateurs passés, pour calculer un score d'adéquation des contraintes à ce modèle. Cette mesure est ensuite employée comme pondération pour un problème d'optimisation sous contraintes afin d'identifier les contraintes divergeant le plus du retour utilisateur pour les enlever en priorité. Des tests sur des jeux de données et des cas pratiques d'utilisation montrent la pertinence de cette approche.

Mots-clés

Clustering sous contraintes incrémentales, programmation par contraintes, insatisfiabilité, ensemble de correction minimal

Abstract

Constraint-incremental clustering consists in letting an expert iteratively add constraints to improve an initial partition step-by-step. Conflicts can occur when a newly added constraint contradicts others given in previous iterations. We address this issue by introducing a hybrid approach combining machine learning and constraint programming (CP). Our method learns a model that scores the adequacy of each constraint with respect to past user feedback. These scores are then used in a CP model computing a minimal correction set, identifying the constraints whose removal would resolve the conflict while preserving those most aligned with user intent. Experiments on synthetic and real-world datasets demonstrate the efficiency and practical relevance of this guided correction mechanism.

Keywords

Constraint-incremental clustering, constraint program-

ming, unsatisfiability, minimal correction sets

1 Introduction

La recherche en IA hybride - l'intégration de différents domaines de l'intelligence artificielle pour créer une synergie entre eux - suscite de plus en plus d'intérêt à l'heure où les demandes d'explicabilité des modèles et d'intégration de l'humain dans les systèmes utilisant l'IA sont des enjeux importants. Parmi les défis posés par ces approches, la gestion du bruit dans le retour utilisateur revêt une importance particulière. Qu'il provienne d'incohérences dans les spécifications de l'utilisateur ou d'interprétations ambiguës, l'expérience montre qu'elles sont inévitables en pratique. En programmation par contraintes (PPC), elles conduisent à des ensembles de contraintes insatisfiables nécessitant une relaxation pour arriver à une solution.

Le clustering est une tâche importante en fouille de données dont l'objectif est de partitionner un ensemble de données (objets) en groupes (clusters) d'une manière telle que les objets appartenant à un même cluster soient similaires mais différents de ceux appartenant aux autres clusters. Le clustering sous contraintes [8] a pour objectif de trouver des clusters plus pertinents en spécifiant, sous forme de contraintes, les propriétés requises. Cela permet de pallier dans une certaine mesure aux biais des algorithmes de clustering. Parmi les contraintes les plus communément utilisées figurent les contraintes *must-link/cannot-link* (ML/CL) imposant que des objets appartiennent (ou non) à un même cluster [21], ou que les clusters aient une taille minimale et/ou maximale [2]. Elles sont en général données par l'expert et modélisent ses connaissances a priori sur le résultat attendu.

Dans la pratique, l'expert peut éprouver des difficultés à spécifier dès le début du processus toutes les connaissances nécessaires à l'élaboration d'une partition intéressante. Une alternative est de lui proposer de réagir sur un résultat en cherchant à le corriger afin de suggérer des connaissances intéressantes à intégrer, conduisant ainsi à une nouvelle partition qui pourra être encore itérativement améliorée. Ceci constitue un processus interactif potentiellement source de

conflits : une nouvelle contrainte peut être incompatible avec une autre proposée plusieurs itérations auparavant, sans que l'utilisateur s'en rende compte.

Résoudre un conflit dans un ensemble de contraintes revient à identifier un sous-ensemble de contraintes à relâcher afin de trouver une solution. Afin de respecter au mieux les désirs de l'utilisateur, ce sous-ensemble doit préserver au mieux les informations déjà données tout en permettant la reprise du processus. Il doit aussi être possible d'expliquer à l'utilisateur comment le choix du sous-ensemble à relâcher a été effectué. Un système qui semble (du point de vue de l'utilisateur) ignorer arbitrairement une partie des informations données peut réduire la confiance dans le processus et décourager l'implication humaine.

Nous proposons une mesure d'adéquation des contraintes exploitant le contexte incrémental, décrivant l'accord entre une contrainte et un modèle d'apprentissage basé sur les retours utilisateurs antérieurs. Nous expliquons également comment utiliser cette mesure en PPC pour trouver un ensemble minimal de correction optimal d'après cette mesure. Nous donnons une implémentation de ces contributions dans un système de clustering sous contraintes incrémentales composé d'un modèle PPC de modification minimale de partition et d'un algorithme de clustering actif reposant sur un arbre de décision.

Dans la section 2, nous passons en revue des travaux antérieurs liés à l'identification de sous-ensembles de contraintes insatisfiables, et au clustering sous contraintes incrémentales. Dans la section 3, nous clarifions le contexte du clustering sous contraintes incrémentales dans lequel s'inscrivent nos contributions. Dans la section 4, nous décrivons la mesure d'adéquation et son utilisation pour le calcul de MCS. Dans la section 5, nous présentons les résultats expérimentaux sur les aspects critiques de cette approche.

2 Travaux antérieurs

2.1 Contraintes et insatisfiabilité

Le problème de satisfiabilité d'un ensemble de contraintes est bien établi en programmation par contraintes et plus largement dans les paradigmes déclaratifs. La recherche sur le sujet est allée au-delà du simple diagnostic d'insatisfiabilité pour développer des méthodes de calcul d'ensembles insatisfiables minimaux (MUS), qui aident à comprendre les causes des conflits en identifiant des combinaisons problématiques de contraintes. De plus, des ensembles de correction minimaux (MCS) peuvent aussi être trouvés ; ils représentent le plus petit ensemble de contraintes à enlever pour rendre le reste satisfiable. Parmi les approches notables, on peut citer CAMUS [14] qui effectue des résolutions successives de problèmes MaxSAT pour énumérer des MCS pour ensuite en déduire des MUS. L'algorithme MARCO [13] parcourt des formules booléennes représentant les différentes parties de l'ensemble de contraintes. Chaque nouvelle formule est testée pour savoir sa satisfiabilité, puis étendue ou rétrécie pour arriver à un MSS ou un MUS selon le résultat. QUICKPLAIN [9] sélectionne un MUS en prenant en compte un ordre donné sur les contraintes pou-

vant refléter des préférences utilisateur. Il fonctionne par recherche dichotomique en respectant l'ordre établi sur les contraintes.

2.2 Clustering sous contraintes incrémentales

Il existe un effort de recherche grandissant dans ce que nous appelons le clustering sous contraintes incrémentale. On peut donner comme point d'origine [6], qui introduit l'idée de récupérer des retours utilisateur de façon itérative à partir d'un résultat existant, au lieu de demander à définir l'ensemble des contraintes à utiliser en une seule fois. Par la suite, d'autres travaux reprendront et étendront ce concept. [19] décrit une méthode basée sur une matrice d'utilité qui représente la valeur de décisions prises selon des attributs catégoriels (par exemple, choisir une place de parking en fonction du bâtiment dans lequel on travaille). Une même décision a donc une valeur différente selon la catégorie à laquelle elle est appliquée. L'objectif est de faire coïncider du mieux possible les clusters avec les catégories qui maximisent une décision particulière. Pour cela, on prend une décision pour chaque cluster de la partition courante et on calcule la distribution des utilités des points selon la décision prise. Des contraintes ML sont générées entre les points qui ont les utilités les plus élevées dans le même cluster, et des contraintes CL sont ajoutées entre les points d'utilités très différentes. On calcule alors une nouvelle partition à l'aide de ces nouvelles contraintes avec COPKMEANS [21]. Dans [16], l'algorithme INCREMENT découpe une partition de départ en sous-clusters et en extrait un représentant de chaque. L'utilisateur étiquette chaque représentant, puis un encodeur apprend un espace sémantique en se basant sur les étiquettes généralisées à l'ensemble des sous-clusters. Enfin, INCREMENT partitionne le nouvel espace en fonction du nombre d'étiquettes distinctes avec KMEANS, ce qui permet de modifier le nombre de clusters de départ tout en propageant les étiquettes dans le jeu de données.

Du côté des approches déclaratives, [10] présente un modèle en programmation linéaire à nombre entiers (PLNE) qui effectue des modifications au niveau des clusters telles que la réduction du diamètre d'un cluster. Une autre approche en PLNE [18] intègre des contraintes en post-traitement à une matrice d'allocation probabiliste des points aux clusters. Le PLNE produit une partition en assignant à chaque point son cluster le plus proche en prenant en compte les contraintes utilisateur de divers types : ML/CL, triplet, taille des clusters, contrainte sur les voisins, les attributs... Ce modèle peut également produire des clusters chevauchants en permettant à un point d'appartenir à plusieurs clusters en même temps.

Plus récemment, [11] présente un système de clustering collaboratif sous contraintes appelé I-SAMARAH. Plusieurs agents calculent différentes partitions d'un jeu de données, puis on identifie les conflits entre les partitions pour les modifier. Quand tous les conflits ont été résolus, un résultat final est désigné par vote. Les contraintes ML/CL sont inté-

grées dans le critère global de qualité de I-SAMARAH, et de nouvelles contraintes peuvent être ajoutées après l'unification. Les contraintes sont dispersées entre les différents agents et servent à provoquer des différences par la création de nouveaux clusters, pour créer de nouveaux conflits et contrebalancer la tendance du système à l'homogénéisation des agents.

L'exploitation de contraintes déjà fournies a été explorée en clustering actif pour générer de nouvelles contraintes à faire valider par l'expert, qui ne sont pas redondantes avec les contraintes connues [22, 1]. Par ailleurs, [20] présente une approche bayésienne pour estimer le meilleur sous-ensemble de contraintes d'un ensemble insatisfiable de contraintes *must-link/cannot-link*. Enfin, [7] décrit deux mesures de qualité d'un ensemble de contraintes : l'informativité par rapport à un algorithme de clustering, c'est-à-dire le taux de contraintes qui ne sont pas immédiatement satisfaites par l'exécution de l'algorithme ; la cohérence des contraintes entre elles selon leur projection de l'une sur l'autre dans un espace métrique, qui estime la redondance de l'ensemble.

3 Clustering sous contraintes incrémentales

Soit $\mathcal{D}$ un jeu de données de n points décrits par m attributs. Le problème du clustering consiste à chercher une **partition** de $\mathcal{D}$ en k clusters de points similaires entre eux et différents des points des autres clusters. Dans le contexte de clustering sous contraintes incrémentales, des informations sur la partition voulue ou des relations entre les points sont exprimées sous forme de contraintes. Ces contraintes sont ajoutées itérativement ; on obtient à chaque itération un nouveau lot de contraintes à utiliser. L'ensemble $\mathcal{C}_t$ contient les contraintes employées à l'itération t pour modifier la partition courante. $\mathcal{C}_t$ peut contenir des contraintes issues des lots précédents. Le lot L_t contient les contraintes obtenues à l'itération t par un algorithme de clustering actif proposant des contraintes informatives et/ou par ajout direct de l'utilisateur. De façon générale, le processus peut être résumé par les étapes suivantes :

1. calculer une partition de départ $\mathcal{P}$;
2. demander à l'expert un lot de contraintes d'après $\mathcal{D}$ et $\mathcal{P}$;
3. modifier $\mathcal{P}$ à l'aide de l'ensemble de contraintes $\mathcal{C}_t$ comprenant le lot de contraintes obtenu L_t et tout ou partie des contraintes antérieures ;
4. répéter les étapes 2 et 3 jusqu'à satisfaction de l'expert ou épuisement des contraintes possibles.

De nombreuses contraintes pour le clustering ont été développées, portant sur les relations entre certains points ou sur des propriétés des clusters eux-mêmes. En l'occurrence, nous nous concentrons sur des contraintes portant sur des paires, des triplets ou des groupes de points.

Définition 3.1 (Contrainte *must-link* [21]). Soit $\mathcal{P}$ une partition d'un jeu de données $\mathcal{X}$. Une contrainte *must-link*

entre deux points x_i et x_j de $\mathcal{X}$ indique qu'ils doivent appartenir au même cluster de $\mathcal{P}$, formellement :

$$ML(x_i, x_j) \stackrel{\text{def}}{=} \forall C_p \in \mathcal{P}, x_i \in C_p \implies x_j \in C_p, \quad (1)$$

Définition 3.2 (Contrainte *cannot-link*). Soit $\mathcal{P}$ une partition d'un jeu de données $\mathcal{X}$. Une contrainte *cannot-link* entre deux points x_i et x_j de $\mathcal{X}$ indique qu'ils doivent appartenir à des clusters différents de $\mathcal{P}$, formellement :

$$CL(x_i, x_j) \stackrel{\text{def}}{=} \forall C_p \in \mathcal{P}, x_i \in C_p \implies x_j \notin C_p, \quad (2)$$

La fermeture transitive sur un réseau de contraintes ML/CL permet d'obtenir des contraintes supplémentaires selon trois règles [15] :

1. $ML(x_a, x_b) \wedge ML(x_b, x_c) \implies ML(x_a, x_c)$;
2. $ML(x_a, x_b) \wedge CL(x_b, x_c) \implies CL(x_a, x_c)$;
3. pour un clustering binaire ($k = 2$), $CL(x_a, x_b) \wedge CL(x_b, x_c) \implies ML(x_a, x_c)$.

Définition 3.3 (Contrainte triplet). Soit $\mathcal{P}$ une partition d'un jeu de données $\mathcal{X}$. Une contrainte triplet entre trois points x_a, x_p et x_n de $\mathcal{X}$ indique que le point x_a est plus similaire à x_p qu'à x_n . Cela implique les relations suivantes en fonction des affectations de $\mathcal{P}$:

$$Triplet(x_a, x_p, x_n) \stackrel{\text{def}}{=} x_p \notin C_a \implies x_n \notin C_a \quad (3)$$

avec C_a le cluster auquel est assigné x_a . Autrement dit, si x_a et x_n sont dans le même cluster, x_p doit également faire partie du cluster puisqu'un point moins similaire à x_a en fait partie.

Définition 3.4 (Plage de clusters d'un ensemble de points). Soit S un ensemble de points d'un jeu de données $\mathcal{X}$. La **plage** (*span*) de clusters de S pour une partition $\mathcal{P}$ est :

$$Sp(S, \mathcal{P}) = \{C_p \mid x \in C_p \cap S\} \forall C_p \in \mathcal{P} \quad (4)$$

Exemple 3.1. La plage de clusters de $\{x_1, x_2, x_3\}$ pour la partition $\{1, 1, 2\}$ est $Sp(\{x_1, x_2, x_3\}, \{1, 1, 2\}) = \{1, 2\}$.

Définition 3.5 (Contrainte sur la plage de clusters). Soit $\mathcal{P}$ une partition d'un jeu de données $\mathcal{X}$ en k clusters. Une contrainte sur la plage de clusters [17] sur $S \subseteq \mathcal{X}$ restreint la plage de clusters $Sp(S, \mathcal{P})$. On distingue deux types de contrainte sur la plage de clusters. Une contrainte spécifique contraint la plage de S par rapport à un ensemble $C \subseteq [1, K]$ donné :

$$SSpan(S, C) \stackrel{\text{def}}{=} Sp(S, \mathcal{P}) \subseteq C \quad (5)$$

Une contrainte sur la plage de clusters générique ne donne qu'une borne supérieure γ sur le nombre de clusters distincts dans S :

$$GSpan(S, \gamma) \stackrel{\text{def}}{=} |Sp(S, \mathcal{P})| \leq \gamma \quad (6)$$

Au fil des itérations, il peut arriver que l'expert donne des contraintes contradictoires avec d'autres reçues à une itération antérieure. Cela peut venir d'une simple erreur, ou d'un changement des attentes au cours du processus pouvant être provoqué par exemple par un déclic en observant les partitions successives. Dans tous les cas, il est nécessaire de résoudre le conflit afin de procéder à la modification et de ne pas bloquer le processus.

3.1 Cadre applicatif

Outre une description théorique, nous proposons une implémentation de notre modèle dans un cadre de clustering sous contraintes incrémentales incorporant un modèle PPC de modification minimale de clustering [3] et un algorithme de clustering actif basé sur les retours utilisateurs passés.

Algorithm 1 Clustering actif sous contraintes incrémentales

Input : Dataset $\mathcal{D}$, budget B
Output : partition $\mathcal{P}$

```

1:  $t \leftarrow 0$ 
2:  $\mathcal{C} \leftarrow \emptyset$ 
3:  $\mathcal{P} \leftarrow \text{initialClustering}(\mathcal{D})$ 
4:  $\mathcal{U}, \text{sat} \leftarrow \text{userFeedback}(\mathcal{P})$  ▷ Retour initial
5: while  $\text{sat} == \text{false}$  do
6:   if  $\mathcal{U} \neq \emptyset$  then
7:      $L_t \leftarrow \mathcal{U}$  ▷ Ajout de contraintes
8:   else
9:      $L_t \leftarrow \text{activeSelection}(\mathcal{D}, \mathcal{P}, B)$  ▷ [4]
10:   $\mathcal{C} \leftarrow \mathcal{C} \cup B_t$ 
11:   $\mathcal{C}_t \leftarrow \text{constraintSet}(\mathcal{C})$ 
12:   $\mathcal{P} \leftarrow \text{modifyClustering}(\mathcal{D}, \mathcal{P}, \mathcal{C}_t)$  ▷ [3]
13:  if  $\mathcal{P} == \emptyset$  then
14:     $\mathcal{P} \leftarrow \text{solveConflict}(\mathcal{C}_t)$  ▷ Section 4
15:   $\mathcal{U}, \text{sat} \leftarrow \text{userFeedback}(\mathcal{P})$ 
16:   $t \leftarrow t + 1$ 
17: return  $\mathcal{P}$ 

```

Ce cadre décrit par l'algorithme 1 repose sur une boucle reposant sur la réaction de l'expert à la partition qui lui est présentée. Il ou elle peut donner ses propres contraintes ou utiliser le clustering actif pour générer des contraintes potentielles à valider. Le lot de contraintes produit ainsi que les contraintes antérieures sont utilisées pour modifier la partition. Notre contribution se place dans le cas où la modification n'est pas réalisable en raison d'un conflit entre les contraintes choisies.

La modification minimale de clustering est faite via un modèle en PPC. La partition modifiée est représentée par des variables entières G_i de domaine $\{1, \dots, k\}$ telle que $G_i = c$ signifie que le point i appartient au cluster c dans la partition modifiée.

$$\min \sum_{i=1}^N \mathbb{I}(\mathcal{P}[i] \neq G_i) d(i, G_i) \quad (7)$$

s.t.

$$Z_j = 1 \iff c_j \quad (8)$$

$$\sum_{c \in \mathcal{C}} Z_c \geq \delta \cdot |\mathcal{C}| \quad (9)$$

avec c_j une contrainte quelconque parmi les types décrits auparavant (ML/CL, triplet ou sur la plage de clusters) et $d(i, G_i)$ la distance du point i à un représentant du cluster G_i auquel il est affecté.

La fonction indicatrice $\mathbb{I}(\mathcal{P}[i] \neq X_i)$ retourne 1 si le numéro de cluster du $i^{\text{ème}}$ élément de la partition modifiée est différent du numéro de cluster à l'entrée du modèle, et 0 sinon.

Les contraintes décrites précédemment sont exprimées de la forme suivante :

- Les contraintes ML/CL sont exprimées par $G_i = G_j$ et $G_i \neq G_j$ respectivement.
- Une contrainte triplet peut s'exprimer avec une implication logique :

$$G_a = G_n \implies G_a = G_p \quad (10)$$

- Les contraintes sur la plage de clusters sont exprimés avec des contraintes globales. Concernant les contraintes spécifiques portant sur un groupe de points S et un ensemble $\mathcal{C}$ de clusters autorisés, des contraintes de domaine peuvent être employées :

$$G_i \in \mathcal{C}, \forall i \in S \quad (11)$$

- Dans le cas des contraintes sur la plage de clusters génériques, on utilise la contrainte globale *atmost_nvalue*¹ [5] :

$$\text{atmost_nvalue}(\gamma, \{G_i \mid i \in S\}) \quad (12)$$

4 Résolution de conflit entre des contraintes de clustering

Pour rappel, un problème de satisfaction de contraintes est posé sur un ensemble de variables $\mathcal{V}$, chacune ayant un domaine de valeurs autorisées dans un ensemble $\mathcal{D}$, et un ensemble de contraintes $\mathcal{C}$ à satisfaire. Un solveur en programmation par contraintes alterne entre la recherche (choisir une variable et y assigner une valeur de son domaine) et la propagation de contraintes (réduire les domaines des autres variables en filtrant les valeurs qui ne permettent plus de satisfaire le problème) jusqu'à trouver une assignation de toutes les variables, solution du problème. Soit un problème de satisfaction de contraintes $\mathcal{T} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ tel que $\mathcal{C}$ est insatisfiable, c'est-à-dire qu'au moins un sous-ensemble $S \subseteq \mathcal{C}$ est tel que ses contraintes ne peuvent pas être toutes satisfaites en même temps. Nous cherchons à résoudre le conflit en identifiant des contraintes à enlever pour permettre la satisfaction de toutes les contraintes restantes. Différents sous-ensembles pertinents peuvent être calculés à partir de ce problème. Leurs définitions sont rappelées ci-dessous [14].

Définition 4.1 (Sous-ensemble insatisfiable minimal (MUS²)). Soit S un sous-ensemble de $\mathcal{C}$. S est un sous-ensemble minimal non satisfiable si et seulement si :

1. https://sofdem.github.io/gccat/gccat/Catmost_nvalue.html
2. De l'anglais *Minimal Unsatisfiable Subset*.

- S est insatisfiable ;
- $\forall c \in S, \{S \setminus c\}$ est satisfiable.

Définition 4.2 (Sous-ensemble satisfaisable maximal (MSS³)). Soit S un sous-ensemble de $\mathcal{C}$. S est un sous-ensemble satisfaisable maximal si et seulement si :

- S est satisfiable ;
- $\forall c \notin S, \{S \cup c\}$ est insatisfiable.

Définition 4.3 (Sous-ensemble de correction minimal (MCS⁴)). Soit S un sous-ensemble de $\mathcal{C}$. S est un sous-ensemble de correction minimal si et seulement si :

- $\mathcal{C} \setminus S$ est satisfiable ;
 - $\forall c \in S, \{\mathcal{C} \setminus \{S \setminus c\}\}$ est insatisfiable.
- De façon équivalente, S est un MCS si et seulement si $\mathcal{C} \setminus S$ est un MSS.

Il y a donc trois façons d'obtenir un MCS : soit par construction depuis l'ensemble vide, soit par énumération des MUS du problème pour trouver un MCS dual, soit par calcul d'un MSS pour obtenir le complémentaire.

4.1 Adéquation d'une contrainte à un modèle

Il existe souvent de multiples façons de résoudre le conflit, de sorte que la recherche d'un MCS est un problème combinatoire. Si l'on recherche un seul ensemble répondant à des exigences pratiques, il faut définir une fonction à optimiser représentant les propriétés désirées. La cardinalité de l'ensemble est l'objectif par défaut dans ce genre de problèmes, mais il est également possible de pondérer les contraintes et chercher le MCS minimisant le poids des contraintes qui le constituent.

C'est dans ce but que nous présentons une mesure destinée à servir de fonction de pondération dans un problème d'optimisation sous contraintes. Cette mesure, que nous appelons adéquation d'une contrainte à un modèle, estime à quel point l'affectation des points d'une contrainte dans un modèle d'apprentissage supervisé entraîné sur des retours utilisateurs passés (étiquetage partiel du jeu de données, contraintes déjà fournies...) est en accord avec la définition de la contrainte. La raison d'être de cette mesure est liée à l'interaction intrinsèque au clustering sous contraintes incrémentales : quand un conflit est détecté dans l'ensemble des contraintes fournies, il faut expliquer à l'utilisateur ce qui pose problème et proposer une solution qui permette de continuer le processus. Dans cet esprit, on considère que - quitte à proposer une résolution du conflit - un MCS pertinent contient des contraintes qui semblent incongrues au vu des informations déjà données par l'utilisateur. On peut alors renforcer l'efficacité de la correction en laissant l'utilisateur confirmer la validité de tout ou partie des contraintes proposées à la relaxation, ou le rejeter pour trouver un autre MCS en écartant ces contraintes, le tout dans un fonctionnement transparent et itératif.

On peut faire le parallèle avec la mesure d'informativité présentée dans [7] : une contrainte informative selon un algorithme de clustering $\mathcal{A}$ contient surtout des contraintes qui

ne sont pas immédiatement satisfaites par la simple exécution de $\mathcal{A}$ sur le jeu de données. De façon analogue, l'adéquation d'une contrainte à un modèle peut être vue comme une mesure de la capacité avec laquelle un modèle d'apprentissage supervisé $\mathcal{M}$ peut satisfaire une contrainte grâce à son seul entraînement.

Le modèle appris est un algorithme d'apprentissage supervisé qui utilise comme étiquettes l'affectation de la partition courante pour l'apprentissage. Son ensemble d'entraînement est un sous-ensemble du jeu de données sur lesquels l'utilisateur a déjà réagi. Ce modèle tente ensuite de généraliser aux points qui ne sont liés à aucune contrainte, représentant ainsi une approximation de la partition désirée par l'utilisateur, d'après les informations connues [4]. En fonction du modèle utilisé (basé sur des arbres, des vecteurs de support, probabiliste...), les paramètres définissant le modèle varient et la façon de calculer l'adéquation change également. Mais la signification pratique est claire : nous voulons éviter d'enlever les contraintes qui répondent parfaitement à cette vision de la partition désirée.

Définition 4.4 (Adéquation d'une contrainte à un modèle). Soit un jeu de données $\mathcal{X}$, un ensemble de contraintes $\mathcal{C}$, $\mathcal{M}$ un modèle d'apprentissage supervisé appris sur un sous-ensemble $\mathcal{X}_C$ de $\mathcal{X}$, contenant les points impliqués dans des contraintes de $\mathcal{C}$, et c une contrainte d'arité n . Quand $\mathcal{M}$ est utilisé pour prédire les assignations du reste du jeu de données, il produit une partition prédit $\mathcal{P}_M$. L'adéquation de c à $\mathcal{M}$ est une fonction $ADQ_M(c) : \mathcal{M} \rightarrow [0, 1]$ telle qu'une valeur de 1 correspond à un accord total avec $\mathcal{M}$ et à l'inverse, une valeur de 0 représente la plus forte dissonance entre $\mathcal{M}$ et la contrainte.

Pour prendre l'exemple des contraintes de clustering définies en Section 3 : une contrainte *must-link*, qui indique que deux points doivent être groupés ensemble, aura une adéquation de 1 si le modèle utilisé les considère comme similaires, et une adéquation de 0 s'ils sont très différents (la notion de similarité dépendant du fonctionnement du modèle utilisé). De même, une contrainte triplet sera d'adéquation maximale si le modèle classe le point de référence et le point positif ensemble tout en classant le point négatif dans un cluster éloigné ou différent ; dans le cas des contraintes sur la plage de clusters, on considère que l'adéquation est nulle si le modèle disperse les points concernés dans plus de clusters que spécifié, sinon on évalue l'adéquation moyenne de chaque sous-ensemble placé dans le même cluster.

On définit les formules suivantes pour chaque type de contrainte afin de calculer leur adéquation à un modèle $\mathcal{M}$ doté d'une fonction de similarité entre deux points $SIM(x, y) : \mathcal{X} \rightarrow [0, 1]$ dépendant du fonctionnement de $\mathcal{M}$:

$$ADQ_M(ML(x, y)) = SIM(x, y) \quad (13)$$

$$ADQ_M(CL(x, y)) = 1 - ADQ_M(ML(x, y)) \quad (14)$$

3. De l'anglais *Maximal Satisfiable Subset*.

4. De l'anglais *Minimal Correction Subset*.

$$ADQ_{\mathcal{M}}(Triplet(a, p, n)) = 0.5 * (1 + \frac{ap - an}{\max(ap, an)}) \quad (15)$$

où $ap = ADQ_{\mathcal{M}}(ML(a, p))$ et $an = ADQ_{\mathcal{M}}(ML(a, n))$. Cette mesure donne 0.5 si le point de référence a est à égale distance de p et de n . Elle augmente à mesure que p se rapproche et/ou n s'éloigne, et diminue s'ils ont le comportement inverse. Dans le cas où les trois points sont confondus, l'adéquation vaut également 0.5, ce qui correspond au but de la fonction : il est étrange de donner une contrainte posant des distances différentes entre trois points s'ils sont confondus.

$$ADQ_{\mathcal{M}}(Span(S, \gamma)) = \begin{cases} 0 & \text{si } Sp(S) > \gamma \\ \sum_{x,y} ADQ_{\mathcal{M}}(ML(x, y)) & \text{sinon} \end{cases} \quad (16)$$

où γ est le nombre de clusters maximal de la contrainte (dans le cas générique) ou la cardinalité de l'ensemble de clusters fourni (dans le cas spécifique).

Les contraintes considérées dans le cadre de cette publication peuvent toutes être retranscrites en un ensemble de contraintes ML, ce qui permet de simplifier les formulations des mesures d'adéquation.

Pour expliciter le principe, prenons un cas concret où le modèle utilisé est un arbre de décision. La fonction SIM peut reposer sur différentes métriques liés aux arbres. Nous choisissons d'utiliser la mesure du dernier ancêtre commun entre deux points (Eq. 17). Ainsi, l'adéquation d'une contrainte ML par rapport à un arbre de décision sera de 1 si les deux points se trouvent dans la même feuille, et va diminuer à mesure que le dernier ancêtre commun est proche de la racine, jusqu'à atteindre 0 s'il s'agit de la racine elle-même. D'autres choix de métriques ou de modèles sont possibles à condition de pouvoir normaliser la mesure. En effet, l'adéquation pouvant porter sur des contraintes de portées très différentes, il est nécessaire de pouvoir exprimer l'adéquation de toute contrainte sur le même intervalle de valeurs afin d'éviter que des contraintes aient un score plus élevée simplement parce qu'elles portent sur un grand nombre de points. Concrètement, cela introduirait un biais favorisant la sélection de contraintes binaires dans la constitution de MCS.

$$LCA(x, y) = \frac{\text{profondeur}(x \cap y)}{\max(\text{profondeur}(x), \text{profondeur}(y))} \quad (17)$$

4.2 Modèle PPC pour choisir le MCS d'adéquation minimale

Afin de calculer un MCS contenant les contraintes d'adéquation minimale à un modèle donné, nous exprimons un problème d'optimisation sous contraintes (COP) en programmation par contraintes en reprenant les variables et contraintes décrites en Section 4. Plus particulièrement, la construction du MCS revient à donner une affectation aux variables binaires Z réifiant les contraintes utilisateur. Dans

une solution du COP, on obtient donc un MCS $\{c_i | Z_i = 0\}$ et un MSS $\{c_i | Z_i = 1\}$. Une fonction objectif pondérant les contraintes avec leur adéquation au modèle appris sur les retours utilisateurs $\mathcal{M}$ oriente le solveur vers le MCS contenant les contraintes les moins adéquates :

$$\arg \min \sum_{i=0}^{|C|} \max(ADQ_{\mathcal{M}}(c), \epsilon) \times \mathbb{I}(Z_i == 0) \quad (18)$$

La fonction indicatrice $\mathbb{I}(Z_i == 0)$ retourne 1 si la contrainte i doit être relâchée (et donc faire partie du MCS) et 0 si elle peut être conservée pour la modification.

Si le poids d'une contrainte est nul, alors l'assignation de la variable binaire correspondante dans le COP n'aura pas d'impact sur l'objectif. C'est pourquoi une valeur ϵ arbitraire non nulle mais proche de 0 est utilisée le cas échéant afin que toutes les contraintes de E soient prises en compte dans le calcul du MCS optimal. Cela est d'autant plus important qu'il paraît très pertinent d'inclure une contrainte avec une adéquation nulle au modèle dans une solution.

Dans le cadre du clustering sous contraintes incrémentales, l'insatisfiabilité d'un CSP à une itération t vient d'au moins une contrainte appartenant au lot L_t qui vient d'être obtenu (dans le cas contraire, la modification au temps $t - 1$ n'aurait pas pu être calculée). Le problème peut donc être restreint à l'ensemble des contraintes contenant au moins un des points impliqués dans les contraintes de L_t :

$$E = \{c \in \mathcal{C}_t \mid \exists l \in L_t \text{ s.t. } \text{portée}(c) \cap \text{portée}(l) \neq \emptyset\} \quad (19)$$

La formalisation du COP s'écrit donc sous la forme suivante :

$$\arg \min \sum_{i=1}^{|E|} w_i (1 - Z_i) \quad (20)$$

$$\text{s.t.} \quad \sum_{j=1}^{|L_t|} (1 - Z_j) \geq 1 \quad (21)$$

$$Z_i \leftrightarrow c_i \quad (22)$$

La contrainte 21 indique qu'au moins une des contraintes du MCS doit provenir du dernier lot de contraintes. Elle sert à orienter la recherche sur un ensemble restreint des contraintes considérées.

5 Expérimentations

Les expérimentations visent à :

- estimer l'impact de la mesure d'adéquation sur les contraintes choisies pour correction.
- déterminer le nombre de contraintes qu'il est possible de traiter en temps raisonnable pour un contexte d'interaction avec l'humain.
- évaluer l'efficacité de la méthode dans un cas d'application réel, sur des séries temporelles d'images satellite.

5.1 Protocole expérimental

Implémentation. Le modèle PPC de calcul de MCS a été implémenté en Python avec CPMpy⁵. Cela permet d'utiliser en même temps le solveur CP-SAT d'or-tools⁶ et les bibliothèques d'apprentissage automatique de Python telles que *scikit-learn*. Nous avons choisi 10^{-4} comme valeur de ϵ ⁷. La prédiction de la partition désirée par l'expert est réalisée avec un arbre de décision de profondeur maximale 3 utilisant l'indice de Gini pour les décisions de branchement.

Méthodologie. Le modèle PPC de calcul de MCS est intégré dans le cadre de clustering actif incrémental sous contraintes décrit en Section 3. L'arbre de décision utilisé pour calculer l'adéquation des contraintes est appris en début d'itération, avant d'obtenir un nouveau lot de contraintes. Le modèle PPC de calcul du MCS optimal est utilisé en *post-hoc*, si la modification de partition échoue. La première itération ne produit pas de conflit : les contraintes sont générées en se basant sur la vérité terrain des jeux de données. Le deuxième lot de contraintes est bruité de façon à introduire un conflit avec le premier lot. Nous avons répété les expériences 10 fois pour prendre en compte la variance des résultats.

Jeux de données. Les expériences ont été réalisées sur trois jeux de données du dépôt de l'UC Irvine⁸ : MNIST (60000 points répartis en 10 classes), Letters (20000 points répartis en 26 classes) et Adult (48842 points répartis en 2 classes).

5.2 Impact de la mesure d'adéquation

Pour évaluer si les MCS calculés en utilisant la mesure d'adéquation sont pertinents, nous comparons les MCS construits à partir d'ensembles de contraintes insatisfiables **sans pondération** (c.à.d le MCS optimal est celui qui contient le moins de contraintes possible), et avec **pondération par l'adéquation** à un arbre de décision appris sur une partition des points soumis aux contraintes.

Nous avons testé différents cas représentant les différentes combinaisons de contraintes possibles parmi celles définies en Section 4 :

- un ensemble de contraintes ML/CL contenant un ou plusieurs cycles contenant une unique contrainte CL [20] ;
- un ou plusieurs triplets en conflit avec des contraintes CL posées sur les mêmes points concernés par le triplet ;
- une contrainte sur la plage de clusters sur un groupe de points impliqués dans un cycle de contraintes CL tel qu'il faut plus de clusters pour satisfaire le cycle de CL que ne le permet la contrainte sur la plage de clusters.

On constate d'après la table 1 que les contraintes consti-

Cas	ML/CL	Triplet	Plage de clusters
MNIST			
Non pondéré	0.67	0.78	0.89
Pondéré	0.45	0.12	0.34
Letters			
Non pondéré	0.6	0.85	0.59
Pondéré	0.34	0.4	0.23
Adult			
Non pondéré	0.89	0.77	0.87
Pondéré	0.24	0.18	0.23

TABLE 1 – Table des valeurs d'adéquation moyennes des contraintes constituant les MCS calculées avec ou sans pondération.

tuant les MCS calculés par notre modèle PPC ont une adéquation plus faible à l'arbre de décision utilisé que celles d'un MCS obtenu sans pondération des contraintes. Quand les MCS calculés avec et sans pondération ont la même cardinalité, le choix optimal fait avec la pondération ne varie pas. A l'inverse, le seul critère de cardinalité introduit de la variance dans les résultats à partir du moment où plusieurs MCS de même taille existent.

5.3 Passage à l'échelle du modèle

Étant donné que le modèle est destiné à une application dans un cadre interactif, il est primordial que la solution optimale soit trouvée en temps réel. Nous avons mesuré le temps de calcul du MCS pour différentes tailles d'ensembles de contraintes ML/CL (de 10 à 10000 contraintes) sur le jeu de données MNIST. La figure 1 montre que le temps de calcul n'excède pas 5 secondes en moyenne pour la plus grande valeur testée. Pour les tailles d'ensembles de contraintes attendues dans le contexte du clustering incrémental sous contraintes, le temps d'exécution est de l'ordre de la seconde.

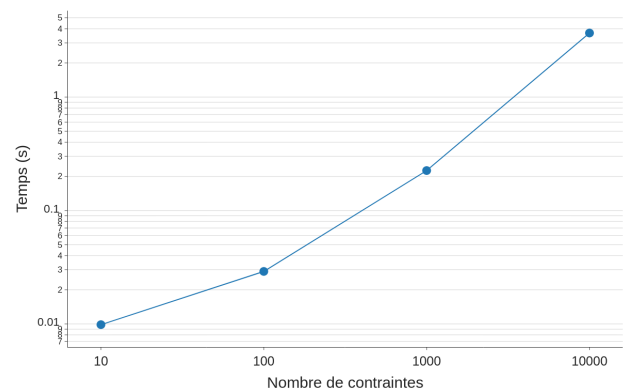


FIGURE 1 – Évolution du temps de calcul de MCS pour différentes tailles d'ensembles de contraintes (échelle logarithmique).

Ces résultats montrent non seulement que ce modèle est

5. <https://github.com/CPMpy/cpm.py>

6. <https://developers.google.com/optimization>

7. Depuis la version 9.24 de CPMpy, des coefficients réels peuvent être utilisés dans la fonction objectif, ce qui évite les transformations en valeurs entières habituellement nécessaires. Nous avons cependant constaté que le solveur cesse de donner des solutions correctes pour des valeurs en-dessous de 10^{-4} .

8. <https://archive.ics.uci.edu/ml/index.php>

suffisamment rapide pour être intégré dans un cadre interactif, mais aussi qu’il peut être employé *avant* de tenter la modification de partition, pour vérifier la satisfiabilité de l’ensemble de contraintes C_t et produire directement un moyen de résoudre le conflit le cas échéant.

5.4 Application sur un cas réel : coupes de bois

Présentation des données. Le jeu de données utilisé est une série temporelle de 11 images satellite de dimensions 724×337 d’une zone des Vosges, prises sur une période de 3 ans (détail en table 2) [12]. Au lieu des canaux habituels, chaque pixel est associé à une série de valeurs NDVI (*Normalized Difference Vegetation Index*) indiquant le niveau de végétation à chaque date.

Période	Date	Coupes
t_1	08/05/2016	-
t_2	23/08/2016	-
t_3	26/08/2016	-
t_4	10/05/2017	8
t_5	19/06/2017	-
t_6	23/08/2017	-
t_7	25/09/2017	-
t_8	08/05/2018	2
t_9	02/07/2018	-
t_{10}	24/07/2018	-
t_{11}	16/08/2018	-

TABLE 2 – Dates des prises de vue de la série temporelle et nombre d’apparitions de zones de coupes claires à chaque période

On dispose d’étiquettes qui décomposent la série temporelle en 3 classes : végétation, artificiel, coupe claire. Cette dernière classe a plusieurs particularités : elle a été construite avec précision par des experts, alors que les deux autres ont été délimitées de façon plus approximative. Elle est aussi très petite en proportion des données totales (moins de 0.3% des données, ce qui correspond à 639 pixels sur un total de 243.988 pixels), ce qui la rend difficile à détecter par des méthodes non supervisées. Elle est composée de 10 zones identifiées par les experts comme des endroits où des arbres ont été coupés. La période d’apparition des zones de coupes claires est indiquée dans la table 2. Enfin, l’évolution temporelle des zones de coupes claires (une baisse forte et ponctuelle du NDVI suivie d’un retour progressif à la normale) est similaire à celle des zones de prairies fauchées ou de récoltes de cultures, ce qui complique encore le problème. Dans ces conditions, l’intervention de l’expert pour dissiper la confusion est primordiale. La figure 2a présente la répartition en pixels des classes annotées et met en exergue la complexité de la tâche.

On dispose de 179 contraintes ML/CL issues de l’expertise humaine (Figure 2b). Pour tester notre méthode, nous introduisons du bruit dans ces contraintes pour simuler des erreurs dans les annotations humaines. Cela se traduit par l’inversion du statut des contraintes bruitées (un

ML devient un CL ou inversement). Plus la proportion de contraintes bruitées est grande, plus l’algorithme d’apprentissage supervisé duquel découle l’adéquation est impacté par le bruit. Nous avons mesuré que notre modèle PPC de calcul de MCS identifie toutes les contraintes bruitées tant que moins 10% de l’ensemble de contraintes est bruité. Au-delà de 25% de contraintes bruitées, les MCS ont autant de chance de comporter de vraies annotations que les contraintes bruitées.

6 Conclusion

Nous avons développé un modèle en programmation par contraintes pour calculer un ensemble de correction minimal sur des ensembles de contraintes en clustering. Ce modèle repose sur une mesure d’adéquation des contraintes à un algorithme d’apprentissage supervisé appris sur les retours utilisateurs passés. Le modèle trouve un MCS optimal efficacement et choisit des contraintes qui semblent contradictoires avec le reste du retour utilisateur. Une piste d’amélioration possible est l’étude du fonctionnement de la mesure d’adéquation avec d’autres modèles (SVM, classificateur bayésien). Des expériences supplémentaires simulant des comportements d’utilisateurs humains dans la génération d’erreurs pourraient confirmer la pertinence de notre méthode.

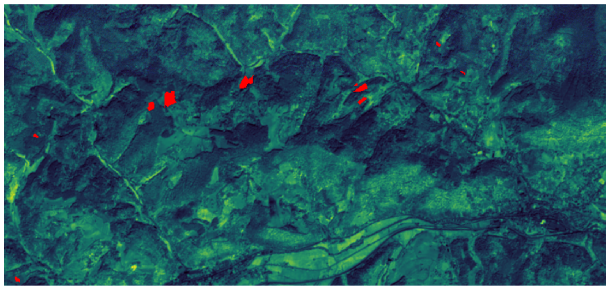
Remerciements

Ce travail est soutenu par le projet ANR HERELLES (Hétérogénéité des données - Hétérogénéité des méthodes : Un cadre collaboratif unifié pour l’analyse interactive de données temporelles)⁹.

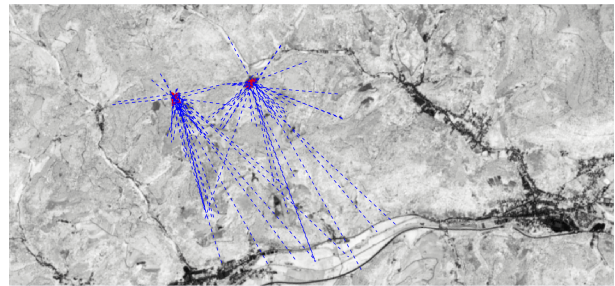
Références

- [1] Ahmad Ali Abin and Hamid Beigy. Active selection of clustering constraints : A sequential approach. *Pattern Recognition*, 47(3) :1443–1458, March 2014.
- [2] Arindam Banerjee and Joydeep Ghosh. Scalable clustering algorithms with balancing constraints. *Data Min. Knowl. Discov.*, 13(3) :365–395, 2006.
- [3] Aymeric Beauchamp, Thi-Bich-Hanh Dao, Samir Loudni, and Christel Vrain. Incremental Constrained Clustering by Minimal Weighted Modification. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10 :1–10 :22, Toronto, Canada, September 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [4] Aymeric Beauchamp, Thi-Bich-Hanh Dao, Samir Loudni, and Christel Vrain. Rule-Based Constraint Elicitation For Active Constraint-Incremental Clustering. In *2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 798–805, October 2024.

9. <https://anr.fr/Projet-ANR-20-CE23-0022>



(a) Zones de coupes de bois sur les images (en rouge).



(b) Mise en évidence des contraintes ML/CL des annotateurs.

FIGURE 2 – Présentation du jeu de données (contraintes ML en rouge, CL en bleu pointillé).

- [5] Christian Bessière, Emmanuel Hébrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Filtering Algorithms for the NValue Constraint. *Constraints*, 11(4) :271–293, December 2006.
- [6] David Cohn, Rich Caruana, and Andrew McCallum. Semi-Supervised Clustering with User Feedback. 4, October 2001.
- [7] Ian Davidson, Kiri L. Wagstaff, and Sugato Basu. Measuring Constraint-Set Utility for Partitional Clustering Algorithms. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Knowledge Discovery in Databases : PKDD 2006*, Lecture Notes in Computer Science, pages 115–126, Berlin, Heidelberg, Germany, 2006. Springer.
- [8] Pierre Gançarski, Thi-Bich-Hanh Dao, Bruno Crémilleux, Germain Forestier, and Thomas Lampert. Constrained Clustering : Current and New Trends. In Pierre Marquis, Odile Papini, and Henri Prade, editors, *A Guided Tour of Artificial Intelligence Research*, pages 447–484. Springer International Publishing, Cham, 2020.
- [9] Ulrich Junker. *QUICKXPLAIN : Preferred Explanations and Relaxations for over-Constrained Problems*. January 2004.
- [10] C.-T. Kuo, S. S. Ravi, T.-B.-H. Dao, C. Vrain, and I. Davidson. A framework for minimal clustering modification via constraint programming. *AAAI*, pages 1389–1395, 2017.
- [11] Baptiste Lafabregue, Pierre Gançarski, Jonathan Weber, and Germain Forestier. Incremental constrained clustering with application to remote sensing images time series. In *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2022.
- [12] Thomas Lampert, Baptiste Lafabregue, Thi-Bich-Hanh Dao, Nicolas Serrette, Christel Vrain, and Pierre Gançarski. Constrained Distance-Based Clustering for Satellite Image Time-Series. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(11) :4606–4621, November 2019.
- [13] Mark H. Liffiton and Ammar Malik. Enumerating Infeasibility : Finding Multiple MUSes Quickly. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874, pages 160–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [14] Mark H. Liffiton and Karem A. Sakallah. Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. *Journal of Automated Reasoning*, 40(1) :1–33, January 2008.
- [15] Pavan Kumar Mallapragada, Rong Jin, and Anil K. Jain. Active query selection for semi-supervised clustering. In *2008 19th International Conference on Pattern Recognition*, pages 1–4, Tampa, FL, USA, December 2008. IEEE.
- [16] Logan Adam Mitchell. *INCREMENT - Interactive Cluster Refinement*. PhD thesis, 2016.
- [17] Nguyen-Viet-Dung Nghiem, Christel Vrain, and Thi-Bich-Hanh Dao. Knowledge integration in deep clustering. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2022, Proceedings, Part I*, volume 13713 of *Lecture Notes in Computer Science*, pages 174–190. Springer, 2022.
- [18] Nguyen-Viet-Dung Nghiem, Christel Vrain, Thi-Bich-Hanh Dao, and Ian Davidson. Constrained Clustering via Post-processing. In *Discovery Science*, pages 53–67, 2020.
- [19] Swapna Raj Prabakara Raj and Balaraman Ravindran. Incremental Constrained Clustering : A Decision Theoretic Approach. In *Trends and Applications in Knowledge Discovery and Data Mining*, volume 7867, pages 475–486. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [20] Jonas Soenen, Sebastijan Dumančić, Toon Van Craenendonck, and Hendrik Blockeel. Tackling Noise in Active Semi-supervised Clustering. In Frank Hutter, Kristian Kersting, Jefrey Lijffijt, and Isabel Valera, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 121–136, Cham, 2021. Springer International Publishing.
- [21] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained K-means Clustering with

Background Knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 577–584, San Francisco, CA, USA, June 2001. Morgan Kaufmann Publishers Inc.

- [22] Aras Yurtman, Wannes Meert, and Hendrik Blockeel. *COBRAS+ : Reusing Previously Obtained Constraints in Active Semi-Supervised Clustering*. November 2021.

Convergence accélérée de la Décomposition Lagrangienne pour la Planification Automatique

Hugo Barral^{1,2}, Quentin Cappart¹, Marie-José Huguet², Sylvie Thiébaux^{2,3}

¹ Polytechnique Montréal, Montreal, Canada

² LAAS-CNRS, Université de Toulouse, INSA, Toulouse, France

³ Australian National University, Canberra, Australie

{hugo.barral,quentin.cappart}@polymtl.ca

huguet@laas.fr

sylvie.thiebaux@anu.edu.au

Résumé

La décomposition lagrangienne est une méthode de programmation linéaire en nombres entiers qui exploite la structure d'un problème pour construire des sous-problèmes indépendants. La résolution de ces sous-problèmes procure une borne à l'optimalité sur le problème original. La qualité de cette borne peut être optimisée par des itérations de sous-gradients sur un vecteur de multiplicateurs. On compte plusieurs contributions qui transposent ces principes en programmation par contraintes. Une approche récente, introduite par Bessa et al. (2025) permet d'inférer une meilleure borne grâce à un modèle d'apprentissage. Notre projet de recherche vise à trouver des transpositions de cette méthode vers d'autres domaines de résolution de problèmes combinatoires et d'en évaluer l'apport. Une première discipline identifiée est la planification automatique. Ce papier montre nos avancées préliminaires pour améliorer une méthode existante de décomposition lagrangienne en planification. Nos premiers résultats témoignent d'une avancée des performances avant d'avoir incorporé l'apprentissage dans notre méthodologie.

Mots-clés

Décomposition lagrangienne, Apprentissage automatique, Planification automatique.

Abstract

Lagrangian decomposition is an integer programming method that exploits the structure of a problem to construct independent sub-problems. Solving these sub-problems provides an optimality bound on the original problem. The quality of the bound can be optimized by subgradient iterations on a vector of multipliers. Many contributions transpose these principles to constraint programming. A recent approach, introduced by Bessa et al. (2025), allows a better bound to be inferred using a learning model. Our research aims to find transpositions of this method to other domains of combinatorial optimization, and to evaluate its contribution. A first discipline identified is AI planning. This paper shows our preliminary advances in improving an existing

Lagrangian decomposition method for planning. Our first results show an improvement in performance before incorporating learning into our methodology.

Keywords

Lagrangian decomposition, Machine learning, AI planning.

1 Introduction

Les méthodes complètes en recherche opérationnelle sont utilisées pour résoudre des problèmes combinatoires et garantir l'optimalité des solutions trouvées. Cependant, lorsqu'elles sont confrontées à des problèmes de grande taille, il devient nécessaire de s'appuyer sur des mécanismes permettant de réduire efficacement l'espace de recherche et de guider la recherche d'une solution. Dans les méthodes de recherche opérationnelle classique, comme la programmation linéaire en nombres entiers (PLNE), une famille de techniques permet de tirer parti de la structure globale du problème et de ses contraintes en le décomposant [28, 23]. La résolution de ces décompositions fournit une borne à l'optimalité sur laquelle la résolution peut s'appuyer pour éliminer une partie des solutions candidates de l'espace de recherche. En programmation par contraintes (PPC), il existe déjà des approches inspirées par la décomposition en PLNE [17] pour utiliser ces méthodes sur un ensemble plus varié de problèmes. L'une d'entre elles, la décomposition lagrangienne [12], a été l'objet de contributions récentes [13, 2] proposant de borner un modèle de PPC ou d'améliorer la propagation d'un groupe de contraintes afin d'accélérer la convergence. La difficulté de cette méthode reste la recherche de multiplicateurs lagrangiens permettant d'obtenir une borne la plus stricte possible. Plusieurs contributions ont essayé de répondre à des problématiques similaires en PLNE avec des modèles d'apprentissage automatique [1, 5, 8] mais ce n'est que plus récemment qu'une méthodologie a été considérée en PPC [3].

Notre projet de recherche vise à prolonger ce travail de transposition en explorant de nouvelles possibilités d'hy-

bridation entre la décomposition lagrangienne et l'apprentissage automatique vers d'autres méthodes de résolution de problèmes combinatoires. Ce papier propose d'analyser une méthode de décomposition existante en planification automatique. Nous détaillons les travaux préliminaires à l'incorporation d'un mécanisme d'apprentissage et les premiers résultats sur les performances. Enfin, nous expliquerons nos futurs travaux pour compléter notre contribution.

2 Méthode initiale

Le papier de Pommerening et al. [22] présente une méthode appliquant la décomposition lagrangienne à la planification automatique. Dans cette section, nous caractérisons les notions de base de la planification et les apports du papier sur lesquels s'appuient nos travaux.

2.1 Notions de base

Une tâche de planification classique peut adopter plusieurs formalismes [11], nous définissons d'abord le système de transitions d'une tâche pour ensuite évoquer SAS⁺ [4].

On définit une tâche de planification par un tuple $\Theta = \langle S, L, c, T, s_0, S_* \rangle$ avec S l'ensemble des états, L l'ensemble des labels des transitions, $c : L \rightarrow \mathbb{R}_0^+$ une fonction de coût, $T \subseteq S \times L \times S$ l'ensemble des relations de transition, s_0 l'état initial et $S_* \subseteq S$ l'ensemble des états terminaux (*goal state*). L'objectif est de trouver un plan correspondant à une séquence de labels $\langle l_1, \dots, l_n \rangle$ permettant de transitionner de s_0 vers un état $s_* \in S_*$. Transitionner d'un état s vers un état s' est possible si $\exists l \in L$ tel que $\langle s, l, s' \rangle \in T$. Chaque état du plan doit pouvoir transitionner vers l'état suivant, tel que $\{\langle s_0, l_1, s_1 \rangle, \langle s_1, l_2, s_2 \rangle, \dots, \langle s_{n-1}, l_n, s_* \rangle\} \subseteq T$. On souhaitera minimiser le coût du plan obtenu par $\sum_{i=1}^n c(l_i)$ pour une recherche optimale.

Le formalisme SAS⁺ permet de représenter ce système de transition de façon plus compacte, via des variables d'état et des actions de changement d'état. On définit une tâche SAS⁺ par un tuple $\Pi = \langle V, O, s_0, s_*, cost \rangle$, avec V l'ensemble des variables, O l'ensemble des actions, s_0 et s_* les assignations partielles de variables de l'état initial et des états terminaux. Finalement, $cost : O \rightarrow \mathbb{R}_0^+$ est une fonction de coût similaire à c . Chaque variable $v \in V$ est caractérisée par un domaine D_v comprenant la valeur indéfinie $\emptyset$. Dans cette représentation, un état s est formalisé par un ensemble de faits. On peut représenter un fait, ou une assignation, par une paire $\langle v, D_v \rangle$. On dénote $vars(s) \subseteq V$ l'ensemble des variables assignées dans s et $s[v]$ la valeur de v dans s . Chaque action $o \in O$ possède des préconditions $pre(o)$ et des effets $eff(o)$, deux ensembles de faits. Une action o est applicable à un état s si $pre(o) \subseteq s$, i.e. tout fait dans $pre(o)$ l'est aussi dans s . La fonction $o(s)$ est l'application de o sur s , pour toutes variables $v \in vars(eff(o))$, $o(s)[v] = eff(o)[v]$, les autres variables restant inchangées. De façon équivalente, l'objectif est de trouver une séquence d'actions $\pi = \langle o_1, \dots, o_n \rangle$ applicable à l'état s_0 et aux états intermédiaires vers un état s_* .

La recherche d'une séquence à travers l'espace d'état est

complexe, d'autant plus lors d'une recherche optimale. Elle est donc nécessairement guidée par une fonction heuristique. L'algorithme de recherche guidée A* [14] priorise l'expansion d'états selon l'évaluation d'une fonction heuristique. On définit $h \in \mathbb{R}_0^+ \cup \{\infty\}$ une fonction heuristique sur un état s et estimant le coût de transition de cet état vers s_* . La valeur ∞ est retournée si l'objectif n'est pas atteignable depuis s . Une heuristique h admissible, i.e., qui ne surestime pas le coût optimal de s vers s_* , permet de guider la recherche tout en préservant son optimalité. On dénote f^* la fonction optimale donnant le coût minimal de s vers s_* , alors h est admissible si $\forall s \in S, h(s) \leq f^*(s, cost)$.

2.2 Heuristiques d'abstraction

En planification, les heuristiques d'abstractions imposent une équivalence entre des états d'un système de transition en les réduisant à un état abstrait. Cette méthode permet de construire un nouveau système de transition dont la taille de l'espace de recherche est réduite et d'en obtenir une valeur heuristique grâce à un algorithme de recherche, comme Dijkstra [9], ou par un solveur grâce à une modélisation en programme de flots. Helmert et al. [16] donnent une définition générale de la représentation d'une abstraction pour un système de transition.

Une abstraction d'un système Θ est une surjection ϵ qui applique S vers un ensemble d'état abstraits S^ϵ . On dénote ensuite le système de transition abstrait Θ_ϵ un tuple $\langle S^\epsilon, L, T^\epsilon, s_0^\epsilon, S_*^\epsilon \rangle$ avec $T^\epsilon = \{(\epsilon(s), l, \epsilon(s')) | (s, l, s') \in T\}$, $s_0^\epsilon = \epsilon(s_0)$ et $S_*^\epsilon = \{\epsilon(s_*) | s_* \in S_*\}$. Le calcul d'une heuristique d'abstraction h_ϵ se fait en deux étapes : l'état courant est d'abord projeté dans l'espace de l'abstraction grâce à la fonction ϵ et on résout le système abstrait en commençant par l'état projeté afin de retourner le coût du plan π_ϵ comme valeur de l'heuristique.

2.3 Partitionnement de coûts

Utiliser plusieurs abstractions différentes d'une tâche est souhaitable pour obtenir une meilleure heuristique par l'addition des résultats des abstractions. Seulement, ces heuristiques sont rarement additives [29]. C'est à dire que la somme de leur valeur peut ne pas être admissible, à cause de la possibilité d'une utilisation redondante d'actions entres plusieurs abstractions. Des contributions proposent de distribuer les coûts de chaque action entre l'ensemble des abstractions pour dépasser cette limite lors de la recherche d'un plan optimal [18]. On note la part des coûts distribués à une abstraction ϵ par un vecteur $cost_\epsilon$. Pour que la distribution assure l'additivité d'un ensemble d'abstractions E , la contrainte suivante doit être satisfaite : $\sum_{\epsilon \in E} cost_{\epsilon,o} \leq cost(o), \forall o \in O$. Les heuristiques de partitionnement de coûts font partie de la famille des heuristiques de comptage d'opérations [21] dont chaque abstraction peut se représenter par un programme linéaire avec des variables $count_{\epsilon,o}$ le nombre d'utilisations de l'action o dans ϵ et des variables auxiliaires y_ϵ . On dénote A_ϵ et B_ϵ des matrices de poids sur le nombre d'utilisation des actions et b_ϵ un vecteur contraignant un nombre d'utilisation minimum sur les ensembles d'actions. La forme générale de ces programmes est la sui-

vante :

$$\begin{aligned} \min \quad & cost_\epsilon^\top count_\epsilon \\ \text{s.t} \quad & A_\epsilon count_\epsilon + B_\epsilon y_\epsilon \geq b_\epsilon \\ & count_\epsilon, y_\epsilon \geq 0 \end{aligned} \quad (1)$$

Pour qu'une heuristique de comptage soit admissible, il faut que chaque contrainte ait une solution pour chaque plan π_ϵ où $count_{\epsilon,o}$ est équivalent au nombre de o dans le plan.

2.4 Décomposition lagrangienne

Pommerening et al. [22] montrent un parallèle entre le partitionnement de coûts et la décomposition lagrangienne en s'appuyant sur la formalisation en programme linéaire de ces heuristiques. La décomposition lagrangienne exploite la structure des problèmes d'optimisation par l'utilisation des variables liantes entre des groupes de contraintes. Elle introduit tout d'abord des contraintes d'équivalence entre les variables liantes et de nouvelles variables indépendantes qui les remplaceront dans les contraintes du problème original. Par exemple pour un programme générique avec des variables x et une série de contraintes $A_i x + B_i y_i \geq b_i$, avec $1 \leq i \leq k$. On peut créer de nouvelles variables x_i , de nouvelles contraintes $x = x_i$ puis remplacer les contraintes sous la forme $A_i x_i + B_i y_i \geq b_i$. La relaxation lagrangienne est alors appliquée aux contraintes d'équivalences en les déplaçant dans la fonction objectif avec un vecteur $\lambda \in \mathbb{R}_{\geq 0}^k$ appelé multiplicateurs lagrangiens. Par exemple, une contrainte générique $Ax \geq b$ relaxée par cette méthode est ajoutée à la fonction sous la forme $\lambda(b - Ax)$. Le choix des multiplicateurs λ est important puisque ces derniers agissent directement sur la qualité de la borne. Dans la littérature, les multiplicateurs peuvent être optimisés grâce à leur dual et par plusieurs méthodes, tels que l'algorithme du sous-gradient qui sera abordé plus loin dans ce papier. L'objectif est d'ensuite pouvoir décomposer le problème initial en séparant les contraintes originales dans ϕ_k en sous-problèmes que l'on peut ensuite formaliser comme suit :

$$\begin{aligned} \phi_i(\lambda_i) = \min \quad & \lambda_i^T x_i \\ \text{s.t} \quad & A_i x_i + B_i y_i \geq b_i \\ & x_i, y_i \geq 0 \end{aligned} \quad (2)$$

Un sous problème supplémentaire ϕ_0 est formalisé pour définir la contrainte $\sum_{i=1}^k \lambda_i \leq c$ sur les multiplicateurs :

$$\begin{aligned} \phi_0(\lambda) = \min \quad & (c - \sum_{i=1}^k \lambda_i)^T x \\ \text{s.t} \quad & x \geq 0 \end{aligned} \quad (3)$$

Les k sous-problèmes montrent une similitude avec les heuristiques d'abstractions partitionnées et leurs représentations mathématiques. Les multiplicateurs lagrangiens sont en fait un équivalent au coût partitionné dans chaque abstraction et les variables x sont similaires aux variables $count$. La contrainte d'admissibilité sur le partitionnement

du coût est assurée par le sous problème supplémentaire. On peut donc employer les mêmes méthodes d'optimisation des multiplicateurs lagrangiens aux coûts partitionnés pour resserrer l'heuristique vers le coût optimal de la tâche de planification. Pommerening et al. [22] itère sur des méthodes de sous-gradient [26] pour améliorer un partitionnement initial. Chaque abstraction est résolue lors d'une itération t et retourne un plan $\pi_\epsilon^{(t)}$ afin de compter le nombre d'occurrences des actions. Le partitionnement des coûts pour l'itération suivante est calculée par sous-gradient, i.e., $cost_\epsilon^{(t+1)} = cost_\epsilon^{(t)} + \eta(t) occurrences(\pi_\epsilon^{(t)})$, avec η une fonction de pas et $occurrences$ une fonction retournant le vecteur d'occurrences des actions dans un plan. Une projection du partitionnement dans un espace admissible est requise en fin d'itération pour satisfaire la contrainte d'additivité. Enfin, la valeur heuristique est retournée par l'addition du coût des plans de chaque abstraction à la dernière itération.

3 Implémentation actuelle

Dans cette section nous décrivons les changements apportés au planificateur *Fast Downward* [15] pour l'implémentation des mécanismes de partitionnement et de décomposition lagrangienne. L'implémentation résultante est ensuite comparée avec celle initialement proposée par Pommerening et al. [22] disponible en ligne ¹.

3.1 Améliorations apportées

Notre première amélioration a consisté à transposer les algorithmes de décomposition lagrangienne vers une version plus récente de Fast Downward (24.06) car plusieurs changements majeurs ont été opérés depuis la version utilisée par Pommerening et al. [22]. Nous avons aussi simplifié le code et amélioré sa consommation mémoire en ne retranscrivant que la partie nécessaire à nos travaux. Par exemple, nous avons fait le choix de ne pas transposer les structures et méthodes qui résolvent les abstractions avec un programme linéaire et de réduire la taille des systèmes de transitions abstraits. Cette démarche nous permet d'économiser de la mémoire vive lorsque nous générons de nombreuses abstractions de nos tâches. L'amélioration n'a pas d'impact sur la structure des algorithmes de recherche, par conséquent, le nombre de nœuds étendus et le nombre de nœuds évalués sont identiques lorsque les paramètres du planificateur sont les mêmes.

Notre deuxième amélioration a été de changer la fonction de pas dans le calcul du sous-gradient. Pommerening et al. [22] implémentent un pas tel que $\eta(t) = \frac{1}{t}$. Cette méthode possède plusieurs désavantages, notamment, elle ne prend pas compte de la possibilité que les coûts entre les actions soient hétérogènes. Un effet de zigzag peut également apparaître sur le gradient, car la méthode le pousse dans une mauvaise direction, e.g., la direction inverse de l'itération précédente. Un effet de "sur place" peut aussi être constaté pendant l'étape de projection. La littérature sur les méthodes de sous-gradients répertorie des approches qui ré-

1. <https://zenodo.org/records/5500499>

duisent ces phénomènes [10]. Nous avons implémenté un nouveau pas inspiré du pas de Polyak [19]. On remplace alors $\eta(t)$ par $\beta^{(t)}(ub_\epsilon^{(t)} - h_\epsilon^{(t)})/\|d^{(t)}\|^2$. $\beta^{(t)} \in [0, 2]$ est un paramètre arbitraire, $ub_\epsilon^{(t)}$ est une estimation de la borne haute de notre heuristique sur l'abstraction ϵ , $h_\epsilon^{(t)}$ est la valeur de notre heuristique d'abstraction ϵ et $d^{(t)}$ le vecteur de la règle de déviation. Nous avons fait le choix de diminuer $\beta^{(t)}$ de moitié lorsque la somme des abstractions n'augmente pas pendant 10 itérations de sous-gradients. La borne $ub_\epsilon^{(t)}$ est calculée sur le plan $\pi_\epsilon^{(t)}$ avec le coût original des actions utilisées. Ainsi, l'aspect hétérogène des actions et de leur répartition à travers les abstractions est pris en compte dans la méthode de sous-gradients. La valeur $d^{(t)}$ est calculée par $d^{(t)} = \alpha occurrences(\pi_\epsilon^{(t)}) + (1 - \alpha)d^{(t-1)}$. $\alpha \in [0, 1]$ est un paramètre arbitraire et $d^0 = \vec{0}$. Le but de $d^{(t)}$ est de "dévier" le gradient avec le calcul de déviation précédent pour diminuer l'effet de zigzag.

3.2 Résultats comparatifs

Nous avons effectué plusieurs expérimentations comparatives entre notre version de la décomposition lagrangienne dans Fast Downward et celle de Pommerening et al. [22]. Nous utilisons les instances de tâches de la piste optimale de l'*International Planning Competition* (IPC) de 1998 à 2018 et nous limitons le temps de recherche à 1800s et la mémoire à 2GB pour chaque instance. La limite de temps correspond au standard d'IPC et la limite de mémoire rejoint celle de Pommerening et al. pour leurs expérimentations. Nous avons paramétré les deux approches afin qu'elles utilisent toutes deux des abstractions de 2 variables au plus et nous nous limitons aux patrons intéressants pour du partitionnement non négatif [20]. Le partitionnement initial est le partitionnement saturé [24] qui était celui avec les meilleures performances initiales dans le papier initial. Le processus de partitionnement est appelé uniquement sur le premier état évalué, les itérations de sous-gradients sont exécutées à chaque état et sont limitées à 200. Les paramètres de notre pas sont $\alpha = 0.5$ et $\beta_1 = 0.001$.

Premièrement, nous avons constaté que les longueurs des plans conjoints aux deux approches ne sont pas différentes, ce qui veut dire que notre approche reste optimale. Une première différence est que notre approche résout 453 tâches sur les 1827 instances sélectionnées contre 427 pour la méthode initiale. La valeur évaluée par notre heuristique sur l'état initial est plus élevée dans 522 tâches. Cet écart montre que notre changement de pas rend notre heuristique plus informative, puisque les deux méthodes utilisent un partitionnement ainsi qu'un nombre d'itérations similaire. Ce changement s'explique par la prise en compte des coûts hétérogènes qui permet des pas plus grands et par mitigation des effets de stagnation lors des itérations. Nous pouvons également constater une baisse de l'empreinte mémoire de notre implémentation, ce qui pourrait s'avérer utile pour des instances ou des configurations générant des abstractions en plus grand nombre. La figure 1 présente une comparaison entre les deux méthodes en termes de nombre de nœuds évalués et de nombre de nœuds étendus. Un point

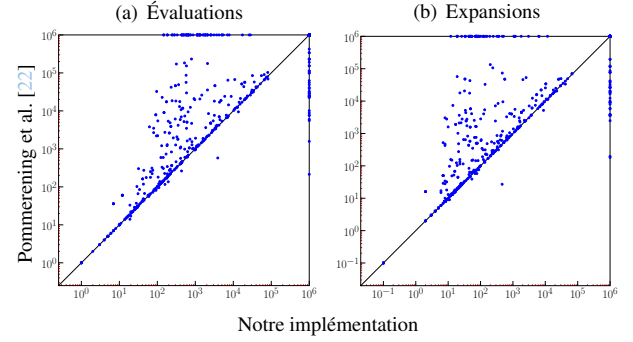


FIGURE 1 – Distribution comparative du nombre de nœuds évalués a et étendus b entre les deux implémentations sur les instances références.

s'écartant d'un côté ou de l'autre de la diagonale indique un nombre supérieur d'évaluations ou d'expansions pour la méthode sur l'axe associé. Le nombre de nœuds évalués est inférieur dans 208 tâches contre 26 et le nombre de nœuds étendus est inférieur dans 218 tâches contre 30 pour notre méthode. Nous pouvons donc étendre et évaluer moins d'états, parfois d'un ordre supérieur à 10^3 , tout en résolvant plus de tâches.

4 Futurs travaux

A terme, notre objectif de recherche est d'obtenir directement des multiplicateurs lagrangiens donnant une borne strict et d'ainsi éviter les étapes de sous-gradients qui sont prohibitives en temps de calcul. A cette fin, notre prochaine étape est de quantifier le potentiel du gain de temps que peut apporter un modèle d'apprentissage des multiplicateurs lagrangiens. Pour cela, nous allons mettre en place une heuristique *oracle* en encodant les évaluations d'état de la décomposition lagrangienne sur nos instances de référence. Une comparaison de cette méthode avec notre implémentation nous permettra d'obtenir la quantité de temps gagné par l'économie des itérations du mécanisme du sous-gradient et du partitionnement des coûts afin de valider notre hypothèse de recherche.

Ensuite, il conviendra de définir notre modèle d'apprentissage. Celui-ci pourra s'inspirer des dernières contributions sur des représentations en graphes dans le domaine de la planification automatique [7, 6] ainsi que des mécanismes d'apprentissage des multiplicateurs lagrangiens dans le domaine de la programmation par contraintes [3]. Il nous faudra également étendre les expérimentations à la comparaison avec d'autres planificateurs de l'état-de-l'art. Un point intéressant sera d'analyser les performances par rapport au planificateur *Scorpion* [25] sur un jeu d'instances communes. Il s'agit d'un planificateur employant également le partitionnement de coûts comme heuristique principale et démontrant d'excellents résultats sur la dernière compétition IPC [27]. Nous voulons enfin pouvoir étendre notre recherche à d'autres problèmes combinatoires en dehors de la planification.

Références

- [1] Ahmed Abbas and Paul Swoboda. DOGE-train : discrete optimization on GPU with end-to-end training. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024.
- [2] David Bergman, Andre A. Cire, and Willem-Jan van Hoeve. Improved constraint propagation via lagrangian decomposition. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming*, pages 30–38, Cham, 2015. Springer International Publishing.
- [3] Swann Bessa, Darius Dabert, Max Bourgeat, Louis-Martin Rousseau, and Quentin Cappart. Learning Valid Dual Bounds in Constraint Programming : Boosted Lagrangian Decomposition with Self-Supervised Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 11113–11121, 2025.
- [4] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4) :625–655, 1995.
- [5] Quentin Cappart, David Bergman, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Augustin Parjadis. Improving variable orderings of approximate decision diagrams using reinforcement learning. *INFORMS Journal on Computing*, 34(5) :2552–2570, 2022.
- [6] Dillon Z Chen, Sylvie Thiébaux, and Felipe Trevizan. Learning domain-independent heuristics for grounded and lifted planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20078–20086, 2024.
- [7] Dillon Z. Chen, Felipe Trevizan, and Sylvie Thiébaux. Return to Tradition : Learning Reliable Heuristics with Classical Machine Learning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34(1) :68–76, May 2024.
- [8] Yanchen Deng, Shufeng Kong, Caihua Liu, and Bo An. Deep attentive belief propagation : Integrating reasoning and learning for solving constraint optimization problems. *Advances in Neural Information Processing Systems*, 35 :25436–25449, 2022.
- [9] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1) :269–271, December 1959.
- [10] Antonio Frangioni, Bernard Gendron, and Enrico Gorgone. On the computational efficiency of subgradient methods : a case study with lagrangian bounds. *Mathematical Programming Computation*, 9 :573–604, 2017.
- [11] Hector Geffner and Blai Bonet. *Classical Planning : Full Information and Deterministic Actions*, pages 15–36. Springer International Publishing, Cham, 2013.
- [12] Monique Guignard and Siwhan Kim. Lagrangean decomposition : A model yielding stronger lagrangean bounds. *Mathematical Programming*, 39 :215–228, 1987.
- [13] Minh Hoàng Hà, Claude-Guy Quimper, and Louis-Martin Rousseau. General bounding mechanism for constraint programs. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming*, pages 158–172, Cham, 2015. Springer International Publishing.
- [14] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2) :100–107, 1968.
- [15] Malte Helmert. The Fast Downward planning system. *J. Artif. Int. Res.*, 26(1) :191–246, July 2006.
- [16] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-Shrink Abstraction : A Method for Generating Lower Bounds in Factored State Spaces. *J. ACM*, 61(3), June 2014.
- [17] John N. Hooker. Chapter 15 - operations research methods in constraint programming. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 527–570. Elsevier, 2006.
- [18] Michael Katz and Carmel Domshlak. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, 174(12) :767–798, 2010.
- [19] B.T. Polyak. Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*, 9(3) :14–29, 1969.
- [20] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From Non-Negative to General Operator Cost Partitioning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), Mar. 2015.
- [21] Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. LP-Based Heuristics for Cost-Optimal Planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 24(1) :226–234, May 2014.
- [22] Florian Pommerening, Gabriele Röger, Malte Helmert, Hadrien Cambazard, Louis-Martin Rousseau, and Domenico Salvagnin. Lagrangian Decomposition for Optimal Cost Partitioning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29(1) :338–347, Jul. 2019.
- [23] Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. The Benders decomposition algorithm : A literature review. *European Journal of Operational Research*, 259(3) :801–817, 2017.

- [24] Jendrik Seipp, Thomas Keller, and Malte Helmert. Narrowing the Gap Between Saturated and Optimal Cost Partitioning for Classical Planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017.
- [25] Jendrik Seipp, Thomas Keller, and Malte Helmert. Saturated cost partitioning for optimal classical planning. *Journal of Artificial Intelligence Research*, 67 :129–167, 2020.
- [26] Naum Zuselevich Shor. *Minimization Methods for Non-Differentiable Functions*. Springer Berlin, Heidelberg, 1985.
- [27] Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 International Planning Competition. *AI Magazine*, 45(2) :280–296, 2024.
- [28] François Vanderbeck and Laurence A. Wolsey. *Reformulation and Decomposition of Integer Programs*, pages 431–502. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [29] Fan Yang, Joseph Culberson, Robert Holte, Uzi Zohavi, and Ariel Felner. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research*, 32 :631–662, 2008.

Graphes de positions : représentations d'alignements et recherche de motifs

Rita-Nathalia Assaf¹, Frédéric Lardeux¹ Frédéric Saubion¹

¹ LERIA, Université d'Angers

prenom.nom@univ-angers.fr

Résumé

Dans cet article nous proposons un formalisme pour décrire des relations d'alignement entre des objets afin de définir un espace de positionnement au sein duquel nous cherchons la présence de motifs particuliers. La modélisation proposée s'appuie sur plusieurs graphes décrivant des aspects complémentaires de cet espace de positionnement. Afin de rechercher des motifs dans ces graphes, et donc d'identifier des structures particulières formées par les objets, nous proposons d'utiliser une approche de résolution par contraintes, s'appuyant sur un algorithme de filtrage existant et permettant de traiter le problème d'isomorphisme de sous-graphes. Des expérimentations nous permettent d'évaluer les avantages/inconvénients des représentations proposées. Une application potentielle qui a motivé ce travail est la recherche de structures de tableaux au sein de documents professionnels.

Mots-clés

Isomorphisme de sous-graphes - Modélisation - Algorithmes de filtrage

Abstract

In this paper, we propose a formalism for describing alignment relationships between objects in order to define a positioning space within which we search for specific patterns. The proposed modelling is based on several graphs that describe complementary aspects of this positioning space. To search for patterns in these graphs, and thus identify specific structures formed by the objects, we propose using a constraint-based solving approach. This approach builds on an existing filtering algorithm and enables us to address the subgraph isomorphism problem. We evaluate the advantages/disadvantages of the proposed representations by means of some experiments. A potential application that motivated this work is the search for table structures in professional documents.

Keywords

Subgraph isomorphism - Modelling - Filtering Algorithms

1 Introduction

La modélisation et l'analyse de relations spatiales entre objets [25] est une préoccupation cruciale notamment pour

la compréhension d'images et pour les systèmes d'information géographique [9]. Sur le plan plus formel, nous pouvons mentionner le système RCC-8 (*Region Connection Calculus*) qui permet un raisonnement qualitatif à partir d'un modèle de représentation spatiale et d'un ensemble d'axiomes [4], en s'appuyant notamment sur les intervalles de Allen [1]. Le raisonnement dans RCC-8 étant NP-complet en général [18], des approches efficaces ont été proposées dans la communauté de la programmation par contraintes pour aborder ces systèmes [16, 8]. En outre, il est intéressant de se rappeler que des travaux fondateurs en programmation par contraintes ont justement été motivés par l'analyse de scènes et d'images [15, 13].

Dans ce travail, nous souhaitons restreindre les relations spatiales à des relations d'alignement et nous focalisons notre problématique sur une recherche de motifs. La motivation principale de cette approche trouve son origine dans des travaux précédents que nous avons menés pour extraire automatiquement des informations à partir de documents structurés, et plus particulièrement de factures [22]. Au sein des diverses techniques utilisées, qui incluent de plus en plus d'approches basées sur les réseaux de neurones [20], des approches basées sur des modélisations utilisant des graphes ont été proposées [7, 19, 3]. La figure 1 illustre comment un document peut être modélisé par un graphe une fois son contenu analysé via des outils classiques (OCR, tokenisation...) qui permettent notamment d'obtenir les positions des différents éléments du document. Toutefois, nombre de ces approches, issues de l'apprentissage profond, nécessitent un entraînement coûteux et des jeux de données extrêmement importants. Cela nous a conduits à proposer une approche à base de programmation par contraintes qui nous a permis de traiter des jeux de données réels de manière efficace, dans un contexte réel de gestion électronique de documents [21]. Dans ce contexte, via une modélisation adéquate, la recherche d'un tableau au sein d'un document peut être ramenée à la recherche d'un sous-graphe (le tableau) dans un graphe (le document).

Le problème d'isomorphisme de sous-graphes (SIP) est connu pour être NP-complet dans le cas général [6]. Des algorithmes dédiés à la résolution des problèmes d'isomorphisme de sous-graphes ont ainsi été proposés depuis de nombreuses années [24, 5]. Actuellement des algorithmes exacts performants, basés sur des approches par fil-

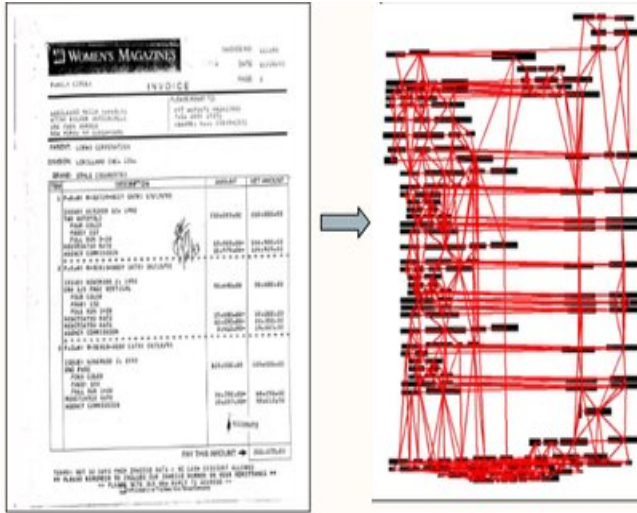


FIGURE 1 – Exemple de transformation d’une facture en un graphe après l’analyse de l’image initiale de la facture en entrée (par exemple un document numérisé). Les arêtes en rouge traduisent les relations d’alignement entre les entités élémentaires extraites. Notons que cette facture contient un tableau (illustration extraite de [3])

trage sont ainsi disponibles [26, 10, 14]. Ces dernières approches s’appuient sur les principes de la programmation par contraintes, notamment en tirant partie d’algorithmes de filtrage spécifiquement dédiés au SIP [27, 23], pouvant être vu alors comme une contrainte globale. En effet, les problèmes d’isomorphisme de sous-graphes peuvent aussi être formulés comme des CSP de manière assez directe, notamment en lien avec des contraintes globales de type *Alldiff* [17].

Dans ce contexte, les objectifs de ce travail exploratoire sont, d’une part de proposer un cadre général pour la modélisation d’espaces de positionnement sous forme de graphes représentant plusieurs points de vue complémentaires (alignement de tokens basiques, organisation en lignes et colonnes...) et, d’autre part, d’étudier comment cette modélisation peut faciliter la recherche de motifs au sein de ces espaces de positionnement. En nous concentrant sur le problème sous-jacent d’isomorphisme de sous graphes, nous nous appuyons sur un algorithme simple de filtrage [27], utilisant les spécificités de notre modélisation. Notre but n’est pas ici de proposer un nouvel algorithme de filtrage mais plutôt d’analyser comment se comportent différents modèles possibles, intégrant une contrainte globale dédiée, sur quelques cas d’études.

L’article est organisé comme suit : après une introduction qui motive l’étude des relations d’alignement entre objets et présente les objectifs et le contexte de ce travail, la section 2 définit un cadre formel pour décrire des espaces de positions et propose plusieurs représentations de ces espaces via des graphes, en précisant également la notion de motif dans chacun des cas et leurs liens avec le problème d’isomorphisme de sous-graphes. La section 3 propose une modélisation de la recherche de motifs sous la forme de problèmes

de satisfaction de contraintes et précise les mécanismes de filtrage pouvant être utilisés dans ce contexte. Enfin une section est consacrée à la présentation d’expérimentations visant à illustrer les différents modèles et concepts introduits.

2 Graphes de positions

2.1 Espaces et représentations

Afin de proposer un cadre général, nous nous basons sur des relations d’ordre partiel qui permettent de définir le positionnement relatif d’objets/items/tokens.

Définition 1 (Relation d’alignement). *Etant donné un ensemble de tokens $\mathcal{T}$, une relation d’alignement $< \subseteq \mathcal{T} \times \mathcal{T}$ est une relation d’ordre partiel telle que $\forall t, t', t'' \in \mathcal{T}, (t < t' \wedge t < t'') \Rightarrow (t' < t'' \vee t'' < t')$*

La condition introduite dans cette définition permet d’interdire la présence d’alignements incohérents de notre point de vue, c’est à dire correspondant à une division incongrue d’une même ligne. Nous considérons ici une relation d’ordre partiel stricte. On se donne à présent deux relations d’alignement définies sur $\mathcal{T} \times \mathcal{T}$: $<_h$ (alignement horizontal) et $<_v$ (alignement vertical). Le triplet $(\mathcal{T}, <_h, <_v)$ sera considéré comme un espace de positionnement qui définit ainsi l’organisation des tokens relativement à leurs alignements respectifs. Nous considérerons que tout élément de $\mathcal{T}$ doit avoir un positionnement relatif par rapport à au moins un autre élément (soit vertical soit horizontal). Si on souhaite relâcher cette contrainte, il suffira de considérer des relations réflexives, ce qui ne change pas fondamentalement la suite.

Notons qu’un espace de positionnement peut ne pas être cohérent relativement aux relations $<_h$ et $<_v$ lorsqu’on les interprète en termes d’alignements verticaux et horizontaux des tokens. La cohérence d’un espace de positionnement peut être définie par le fait qu’il n’existe pas quatre tokens $t_1, t_2, t_3, t_4 \in \mathcal{T}$ tels que $t_1 <_h t_2 \wedge t_3 <_h t_4 \wedge t_1 <_v t_4 \wedge t_2 <_v t_3$ ou $t_1 <_v t_2 \wedge t_3 <_v t_4 \wedge t_1 <_h t_3 \wedge t_4 <_h t_2$. Dans la suite, nous ne manipulerons que des espaces de positionnement cohérents.

Notons ici que les graphes grilles (*grid or lattice graphs*) pourraient partager certaines similarités avec notre approche mais ces graphes sont en général réguliers et considèrent comme sommets des points de coordonnées entières et comme arêtes les paires de sommets partageant une coordonnée. Ainsi, nous cherchons ici une représentation plus spécifique et plus déclarative basée sur les alignements.

Afin d’illustrer les différentes représentations que nous allons proposer, nous considérons l’exemple suivant.

Exemple 1. Soit $\mathcal{T} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$. Nous définissons notre espace de positionnement en posant : $1 <_h 2, 2 <_h 3, 3 <_h 4, 5 <_h 6, 6 <_h 7, 8 <_h 9, 9 <_h 10, 1 <_v 5, 5 <_v 8, 2 <_v 6, 6 <_v 9, 9 <_v 11, 3 <_v 7, 7 <_v 10$. Dans cet exemple, nous pouvons observer que l’élément 4 n’est aligné avec aucun autre élément selon $<_v$, ce qui mettra en avant une colonne avec un seul élément. Il en va de même pour l’élément 11 relativement à $<_h$.

Définition 2 (Graphe de positions). *Étant donné un espace de positionnement $\Pi = (\mathcal{T}, <_h, <_v)$, on définit le graphe de positions induit GP_Π comme un graphe étiqueté orienté $(\mathcal{T}, \mathcal{A}, \mathcal{E})$ où $\mathcal{A} \subseteq \mathcal{T} \times \mathcal{T}$ et $\mathcal{E} : \mathcal{T} \times \mathcal{T} \rightarrow \{h, v\}$ tels que :*

- $\forall t, t' \in \mathcal{T}, t <_h t' \Leftrightarrow (t, t') \in \mathcal{A} \wedge \mathcal{E}((t, t')) = h$
- $\forall t, t' \in \mathcal{T}, t <_v t' \Leftrightarrow (t, t') \in \mathcal{A} \wedge \mathcal{E}((t, t')) = v$

Pour l'exemple 1, nous obtenons le graphe de la figure 2. Sur la figure 2, nous ne reportons pas les étiquettes des arcs car les arcs étiquetés 'h' seront horizontaux et naturellement les arcs étiquetés 'v' verticaux.

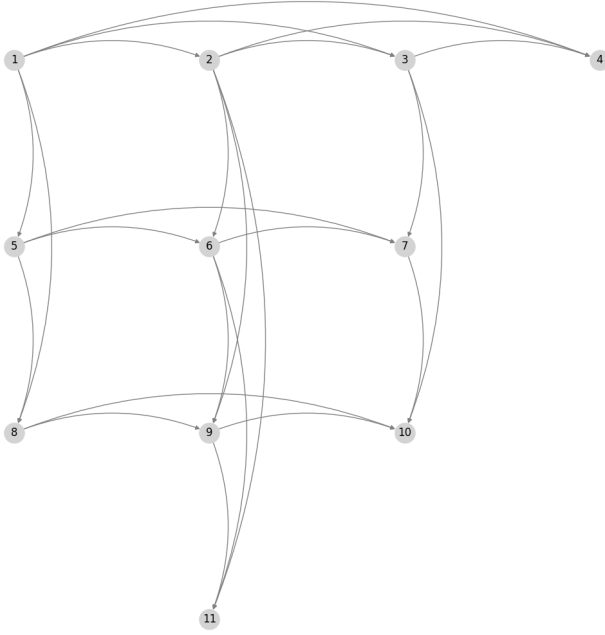


FIGURE 2 – Graphe de positions GP

Notons que les relations d'ordre partiel $<_h$ et $<_v$ sont transitives mais, à l'instar de ce qui est fait pour les diagrammes de Hasse, nous pouvons simplifier notre représentation en ne représentant pas la transitivité dans le graphe de positions.

Définition 3 (Graphe de positions réduit). *Étant donné un graphe de positions $GP_\Pi = (\mathcal{T}, \mathcal{A}, \mathcal{E})$ construit sur un espace de positionnement $\Pi = (\mathcal{T}, <_h, <_v)$, le graphe de positions réduit $GP_\Pi^{red} = (\mathcal{T}, \mathcal{A}^{red}, \mathcal{E}^{red})$ est obtenu à partir de GP_Π avec*

$$\mathcal{A}^{red} = \mathcal{A} \setminus \{(t, t'') \in \mathcal{A} \mid \exists t' \in \mathcal{T} (t, t') \in \mathcal{A} \wedge (t', t'') \in \mathcal{A}\}$$

$\mathcal{E}^{red}$ est naturellement la restriction de $\mathcal{E}$ à $\mathcal{A}^{red}$

Nous avons considéré des relations d'ordre strict, si toutefois on prenait en compte un ordre réflexif, il conviendrait de retirer également l'ensemble $\{(t, t) \in \mathcal{A}\}$. Toujours avec l'exemple 1, le graphe réduit est représenté sur la figure 3.

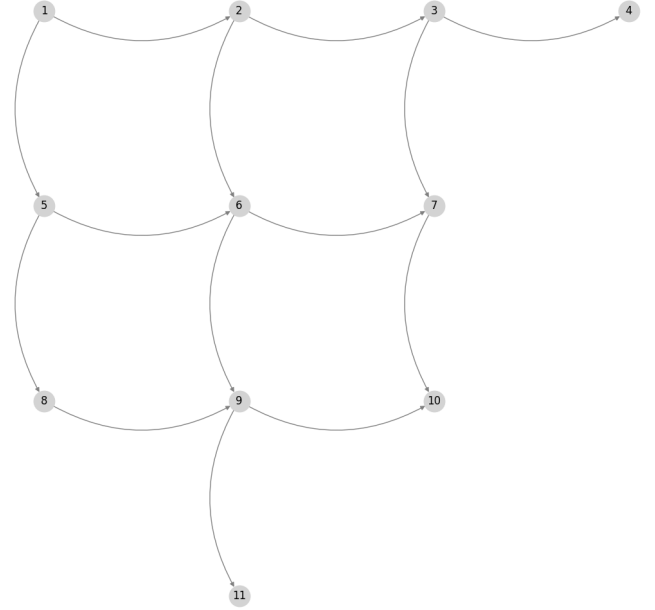


FIGURE 3 – Graphe de positions réduit GP^{red}

En utilisant les relations d'alignements $<_h$ (resp. $<_v$), nous définissons de manière naturelle la notion de ligne (resp. colonne).

Définition 4 (Lignes et colonnes). *Étant donné un espace de positionnement $\Pi = (\mathcal{T}, <_h, <_v)$, une ligne $L \subseteq \mathcal{T}$ (resp. une colonne $C \subseteq \mathcal{T}$) est un sous-ensemble totalement ordonné $(L, <_h)$ (resp. $(C, <_v)$). $\mathcal{L}_\Pi \subseteq 2^\mathcal{T}$ (resp. $\mathcal{C}_\Pi \subseteq 2^\mathcal{T}$) est l'ensemble des lignes (resp. des colonnes) associé à Π .*

Remarquons que $(\mathcal{L}_\Pi, \subseteq)$ et $(\mathcal{C}_\Pi, \subseteq)$ sont deux ensembles partiellement ordonnés selon la relation d'inclusion. Une ligne (resp. colonne) complète sera un élément maximal de $(\mathcal{L}_\Pi, \subseteq)$ (resp. $(\mathcal{C}_\Pi, \subseteq)$). Notons que l'ensemble des lignes (resp. colonnes) complètes $\mathcal{L}_\Pi^{comp}$ (resp. $\mathcal{C}_\Pi^{comp}$) forment une partition de $\mathcal{L}_\Pi$ (resp. $\mathcal{C}_\Pi$).

Définition 5 (Graphe des lignes/colonnes). *Étant donné un espace de positionnement $\Pi = (\mathcal{T}, <_h, <_v)$, $GLC_\Pi = (\mathcal{L}_\Pi \cup \mathcal{C}_\Pi, \mathcal{I})$ est le graphe non-orienté¹ des lignes/colonnes associées à Π , tel que $\mathcal{I} \subseteq \mathcal{L}_\Pi \times \mathcal{C}_\Pi$ vérifie $\forall (l, c) \in \mathcal{I}, l \cap c \neq \emptyset$. GLC_Π est un graphe biparti.*

Notons qu'il existe une arête du graphe non orienté GLC_Π , notée (l, c) qui correspond en fait à chaque token $t \in \mathcal{T}$ tel que $t \in c \wedge t \in l$. Le graphe de lignes/colonnes obtenu pour l'exemple 1 correspond à la figure 4. Nous pouvons remarquer que ce graphe possède considérablement plus d'arêtes

1. Comme nous allons manipuler des graphes orientés et non orientés, afin de pas alourdir le vocabulaire et les notations, nous noterons ici $(l, c) \in \mathcal{I}$ le fait que l'arête (l, c) est dans $\mathcal{I}$, de la même manière que pour les arcs. Le contexte, en général clair, permettra de distinguer cette nuance habituelle.

que le graphe de positions. Toutefois, ayant une structure bipartie, cela ne permet pas forcément de présumer de la difficulté de trouver un motif en son sein.

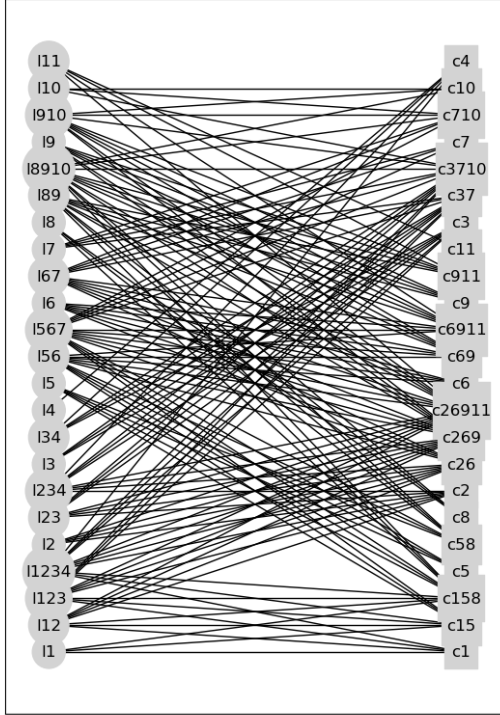


FIGURE 4 – Graphe GLC - Une ligne $\{5, 6\}$ est notée 156, une colonne $\{3, 7\}$ c37

Etant donné un graphe de lignes/colonnes $GLC_{\Pi} = (\mathcal{L}_{\Pi} \cup \mathcal{C}_{\Pi}, \mathcal{I})$, le graphe réduit de lignes/colonnes $GLC_{\Pi}^{red} = (\mathcal{L}_{\Pi}^{comp} \cup \mathcal{C}_{\Pi}^{comp}, \mathcal{I}^{comp})$ est obtenu en restreignant $\mathcal{L}_{\Pi}$ (resp. $\mathcal{C}_{\Pi}$) aux lignes (resp. colonnes) complètes et en réduisant naturellement $\mathcal{I}^{comp}$.

Remarque : Comme les lignes et colonnes sont des sous-ensembles de $\mathcal{T}$, elles pourraient être également vues comme des hyperarêtes d'un hypergraphe sur $\mathcal{T}$. Nous ne développons pas cette autre modélisation ici.

Nous allons à présent établir des équivalences entre nos différentes représentations d'un même espace de positionnement.

Propriété 1 (Equivalence entre GP_{Π} et GLC_{Π}). *Etant donné un espace de positionnement $\Pi = (\mathcal{T}, <_h, <_v)$ et $GP_{\Pi} = (\mathcal{T}, \mathcal{A}, \mathcal{E})$, $GLC_{\Pi} = (\mathcal{L}_{\Pi} \cup \mathcal{C}_{\Pi}, \mathcal{I})$ ses représentations associées. Les propositions suivantes sont équivalentes, pour tout couple $(t, t') \in \mathcal{T} \times \mathcal{T}$:*

1. $t <_h t'$ (resp. $t <_v t'$)

2. $(t, t') \in \mathcal{A}$ et $\mathcal{E}((t, t')) = h$ (resp. $\mathcal{E}((t, t')) = v$)
3. $\exists l \in \mathcal{L}_{\Pi}, t \in l \wedge t' \in l$ (resp. $\exists c \in \mathcal{C}_{\Pi}, t \in c \wedge t' \in c$)

Nous noterons la relation d'équivalence induite entre les représentations par $\sim_{\Pi}$, i.e., $GP_{\Pi} \sim_{\Pi} GLC_{\Pi}$

Démonstration. L'équivalence entre 1. et 2. est assurée par la définition 2. Pour montrer l'équivalence entre 1. et 3., il faut utiliser la définition 4 des lignes et des colonnes. Nous avons en effet $t <_h t'$ (resp. $t <_v t'$) qui implique qu'il existe une ligne (resp. colonne) l (resp. c) qui contient t et t' . La réciproque est également évidente au vu de la définition des lignes et colonnes. $\square$

2.2 Isomorphisme de sous-graphes de positions

Afin d'alléger les notations, l'espace de positionnement $\Pi = (\mathcal{T}, <_h, <_v)$ sera supposé connu dans la suite et nous omettons donc les indices Π .

Définition 6 (Motif de positions). *Etant donné un ensemble T , un motif de positions est un graphe dirigé étiqueté $MP = (T, A, E)$, tel que $A \subseteq T \times T$ et $E : A \rightarrow \{h, v\}$.*

Définition 7 (Isomorphisme de sous-graphes de positions). *Soit un graphe de positions $GP = (\mathcal{T}, \mathcal{A}, \mathcal{E})$ et un motif de positions $MP = (T, A, E)$. MP est isomorphe à un sous-graphe de GP si et seulement si il existe une injection $f : T \rightarrow \mathcal{T}$ vérifiant :*

- $(t, t') \in A \Rightarrow (f(t), f(t')) \in \mathcal{A}$
- $(t, t') \in A \Rightarrow E((t, t')) = \mathcal{E}((f(t), f(t')))$

Chaque injection f définit donc un sous-graphe $f(MP) = (f(T), \mathcal{A}_{f(T)}, \mathcal{E}_{f(T)})$ de GP , où $\mathcal{A}_{f(T)} = \{(t, t') \in \mathcal{A} | t, t' \in f(T)\}$ et $\mathcal{E}_{f(T)}$ est la restriction de $\mathcal{E}$ à $f(T)$. On notera alors $GP \triangleright_f MP$.

Sur l'exemple 1, nous pouvons rechercher, dans le graphe GP^{red} , le motif $M = (T = \{a, b, c, d\}, A = \{(a, b), (c, d), (a, c), (b, d)\}, E)$ avec $E((a, b)) = E((c, d)) = h$ et $E((a, c)) = E((b, d)) = v$. Nous aurons alors une solution illustrée par la figure 5.

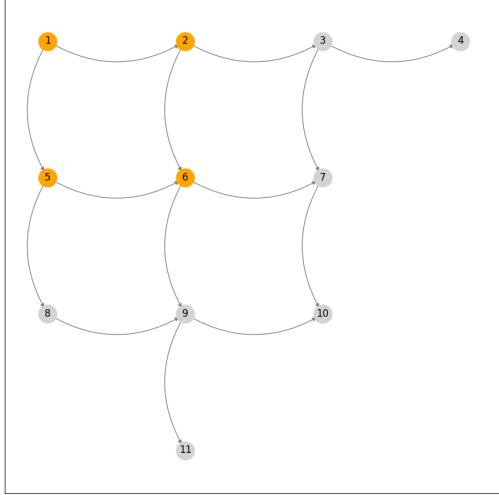
Comme nos représentations sont équivalentes, nous allons étendre cette équivalence à la recherche de motifs. Nous définissons donc la notion de motif de lignes/colonnes.

2.3 Isomorphisme de sous-graphes lignes/colonnes

Définition 8 (Motif de lignes/colonnes). *Un motif de lignes/colonnes est un graphe non orienté $MLC = (TL \cup TC, I)$ défini sur deux ensembles TL, TC tels que $TL \cap TC = \emptyset$ et tel que $I \subseteq TL \times TC$.*

Définition 9 (Isomorphisme de sous-graphes lignes/colonnes). *Soit un graphe de lignes/colonnes $GLC = (\mathcal{L} \cup \mathcal{C}, \mathcal{I})$ et un motif de lignes/colonnes $MLC = (TL \cup TC, I)$. MLC est isomorphe à un sous-graphe de GLC si et seulement si il existe une injection $f : TL \cup TC \rightarrow \mathcal{L} \cup \mathcal{C}$ vérifiant :*

- $\forall l \in TL, f(l) \in \mathcal{L}, \forall c \in TC, f(c) \in \mathcal{C}$
- $\forall (l, c) \in I, (f(l), f(c)) \in \mathcal{I}$


 FIGURE 5 – Motif M sur GP^{red} (en orange)

Chaque injection f définit donc un sous-graphe $f(MLC) = (f(L) \cup f(C), \mathcal{I}_{|f(L) \cup f(C)})$ de GLC où $\mathcal{I}_{|f(L) \cup f(C)} = \{(f(l), f(c)) \in \mathcal{I}\}$. On notera alors $GLC \triangleright_f MLC$.

Propriété 2 (Equivalence de la recherche de motifs). Soit un graphe de positions $GP = (\mathcal{T}, \mathcal{A}, \mathcal{E})$, et un graphe de lignes/colonnes $GLC = (\mathcal{L} \cup \mathcal{C}, \mathcal{I})$, tels que $GP \sim GLC$. Soit un motif de positions $MP = (T, \mathcal{A}, E)$ et un motif de lignes/colonnes $MLC = (TL \cup TC, I)$, tels que $TL \subseteq 2^T$ et $TC \subseteq 2^T$. Il existe une injection fP telle que $GP \triangleright_{fP} MP$ si et seulement si il existe une injection fLC telle que $GLC \triangleright_{fLC} MLC$ qui vérifient :

1. $\forall ml \in TL, \forall t \in T, (t \in ml) \Leftrightarrow fP(t) \in fLC(ml)$
2. $\forall mc \in TC, \forall t \in T, (t \in mc) \Leftrightarrow fP(t) \in fLC(mc)$
3. $\forall ml \in TL, \forall mc \in TC, \forall t \in T, ml \cap mc = t \Leftrightarrow fLC(ml) \cap fLC(mc) = fP(t)$

Démonstration. Supposons qu'il existe une injection $fP : T \rightarrow \mathcal{T}$ telle que $GP \triangleright_{fP} MP$. Soient $TL \subseteq 2^T$ et $TC \subseteq 2^T$ et une fonction $fLC : TL \cup TC \rightarrow \mathcal{L} \cup \mathcal{C}$ qui vérifie les trois conditions 1. 2. et 3. Soient $lc, lc' \in TL \cup TC$ telles que $lc \neq lc'$. Comme fP est injective et avec les conditions 1. et 2., on a forcément $fLC(lc) \neq fLC(lc')$, et donc fLC injective. Par ailleurs, nous savons que le motif $MP = (T, \mathcal{A}, E)$ est isomorphe à un sous-graphe de GP noté $fP(MP)$ (voir définition 7). Comme $GP \sim GLC$, en utilisant la propriété 1, et les conditions 1., 2. et 3. liant fP et fLC , nous pouvons en déduire qu'il existe un sous-graphe de GLC construit comme $(\{l \in \mathcal{L} | \exists fP(t) \in \mathcal{T}, fP(t) \in l\} \cup \{c \in \mathcal{C} | \exists fP(t) \in \mathcal{T}, fP(t) \in c\}, \{(l, c) \in$

$\mathcal{L} \times \mathcal{C} | \exists fP(t) \in \mathcal{T}, fP(t) \in l \cap c\}$ qui est isomorphe au motif de lignes/colonnes $(TL \cup TC, I)$ de telle sorte que $GLC \triangleright_{fLC} MLC$. La réciproque peut être obtenue en suivant le même raisonnement du fait des conditions imposées dans la propriété 2. $\square$

Le corollaire suivant découle de la propriété 2 et de la définition des équivalences entre modélisations de la définition 7.

Corollaire 1. Soit un graphe de positions GP et un graphe de lignes/colonnes GLC tels que $GP \sim GLC$, un motif de positions MP et un motif de lignes/colonnes MLC . S'il existe des injections fP et fLC telles que $GP \triangleright_{fP} MP \wedge GLC \triangleright_{fLC} MLC$ et qu'elles vérifient les conditions de la propriété 2, on a alors $fP(MP) \sim fLC(MLC)$

Ce corollaire nous permet d'établir qu'il est équivalent de rechercher un motif sur le graphe de position GP ou sur son graphe de lignes et colonnes associé, GLC .

La propriété 2 et son corollaire assurent une cohérence entre les motifs. Toutefois, il convient de remarquer que, d'un point de vue opérationnel, étant donné un motif de position MP , il faut construire un motif de lignes/colonnes correspondant MLC . Etant donné $GP \sim GLC$, nous aurons alors un ensemble de solutions $\mathcal{S}_{MP}^{GP} = \{fP | GP \triangleright_{fP} MP\}$ et un ensemble de solutions $\mathcal{S}_{MLC}^{GLC} = \{fLC | GLC \triangleright_{fLC} MP\}$ tels que $\forall fP \in \mathcal{S}_{MP}^{GP}, \exists fLC \in \mathcal{S}_{MLC}^{GLC}, fP(MP) \sim fLC(MLC)$ et réciproquement. Toutefois, les ensembles de solutions ne sont pas directement équivalents. Revenons à notre exemple 1.

Le motif illustré par la figure 5 et défini sur $T = \{a, b, c, d\}$ va nous amener à rechercher un motif MLC défini sur $TL = \{\{a, b\}, \{c, d\}\}$ et $TC = \{\{a, c\}, \{b, d\}\}$ avec $I = \{(\{a, b\}, \{a, c\}), (\{a, b\}, \{b, d\}), (\{c, d\}, \{a, c\}), (\{c, d\}, \{b, d\})\}$. Si on considère la solution $fP = \{a \mapsto 1, b \mapsto 2, c \mapsto 5, d \mapsto 6\}$ présentée sur la figure 5, il existe donc plusieurs solutions pour ce motif dans le graphe GLC . Celle qui nous intéresse est $fLC = \{\{a, b\} \mapsto \{1, 2\}, \{c, d\} \mapsto \{5, 6\}, \{a, c\} \mapsto \{1, 5\}, \{b, d\} \mapsto \{2, 6\}\}$, mais $fLC' = \{\{a, b\} \mapsto \{5, 6\}, \{c, d\} \mapsto \{1, 2\}, \{a, c\} \mapsto \{2, 6\}, \{b, d\} \mapsto \{1, 5\}\}$ est également une solution au problème de recherche de motif. Toutefois cette solution fLC' ne vérifie pas les conditions de la propriété 2 et nous n'avons donc pas $fP(MP) \sim fLC'(MLC)$. Nous avons en effet perdu dans GLC la représentation directe de l'information sur les ordres entre tokens (positionnements relatifs verticaux et horizontaux) en passant d'un graphe orienté à un graphe non orienté.

Nous pouvons toutefois ajouter des contraintes lors de la recherche de motifs dans le graphe de lignes colonnes afin d'extraire de l'ensemble des solutions obtenues celles qui coïncident avec le motif de position initial. Pour des raisons de place, nous ne détaillons pas la formalisation de ce raffinement des solutions mais nous allons décrire de manière opérationnelle comment dans notre recherche de motifs nous prendrons en compte ces contraintes.

3 Recherche de motifs

Dans cette section, nous allons présenter une modélisation de la recherche de motifs sous la forme de CSPs et décrire également un algorithme de filtrage qui permet de traiter spécifiquement le problème d'isomorphisme de sous-graphes.

3.1 Motifs de positions

Soient un graphe $GP = (\mathcal{T}, \mathcal{A}, \mathcal{E})$ et un motif associé $MP = (T, A, E)$ tels que présentés dans la définition 7.

Nous introduisons, de manière naturelle en nous appuyant sur cette définition, le problème de satisfaction de contraintes $(\mathcal{X}, D, \mathcal{C})_P$ (variables, domaines, contraintes) relatif à la recherche de MP dans GP de la manière suivante. L'ensemble des variables de décision est défini par :

$$\mathcal{X} = \{u_t | t \in T\}$$

Nous associons donc à chaque élément du motif recherché une variable de décision dont le domaine est défini par :

$$\forall u_t \in \mathcal{X}, D(u_t) = \mathcal{T}$$

Il convient ensuite de faire le lien entre les arcs de GP et les arcs du motif.

$$\forall u_t, u_{t'} \in \mathcal{X}, (t, t') \in A \Rightarrow (u_t, u_{t'}) \in \mathcal{A}$$

Il reste à poser une contrainte également sur le type des arcs correspondant à chacune des relations d'alignement.

$$\forall u_t, u_{t'} \in \mathcal{X}, E((t, t')) = \mathcal{E}((u_t, u_{t'}))$$

Enfin, comme ce CSP modélise la recherche de la fonction fP de la définition 7 qui est injective, nous ajoutons une contrainte globale bien connue [17] sur l'ensemble des variables de décision $Alldiff(\mathcal{X})$.

3.2 Motifs de lignes/colonnes

Nous considérons à présent un graphe de lignes/colonnes $GLC = (\mathcal{L} \cup \mathcal{C}, \mathcal{I})$ et un motif de lignes/colonnes $MLC = (TL \cup TC, I)$ associé. De la même manière que précédemment, nous définissons un CSP $(\mathcal{X}, D, \mathcal{C})_{LC}$ relatif à la recherche de MLC dans GLC . De la même manière et pour modéliser la fonction fLC de la définition 9, nous posons donc :

$$\begin{aligned} \mathcal{X} &= \{u_{lc} | lc \in TL \cup TC\} \\ \forall u_{lc} \in \mathcal{X}, lc \in TL &\Rightarrow D(u_{lc}) = \mathcal{L} \\ \forall u_{lc} \in \mathcal{X}, lc \in TC &\Rightarrow D(u_{lc}) = \mathcal{C} \\ \forall u_{lc}, u_{lc'} \in \mathcal{X}, (lc, lc') \in I &\Rightarrow (u_{lc}, u_{lc'}) \in \mathcal{I} \\ Alldiff(\mathcal{X}) \end{aligned}$$

Notons bien que le graphe GLC étant non orienté la notation (lc, lc') désigne une paire de valeurs non ordonnée (comme déjà précisé). De plus, le fait que le graphe GLC soit biparti relativement à la partition $\mathcal{L} \cup \mathcal{C}$ de ses sommets nous permet de restreindre la définition des domaines de chaque variable. En lien avec les remarques de la fin de la

section 2.3, nous introduisons des contraintes supplémentaires afin de mieux établir les liens entre motifs de positions et motifs de lignes/colonnes.

Nous définissons deux relations d'ordre sur les lignes et les colonnes induites naturellement par les deux relations d'alignement $<_h$ et $<_v$ en premier lieu sur les lignes et colonnes complètes :

$$\begin{aligned} \forall l, l' \in \mathcal{L}^{comp}, l <_v l' &\Leftrightarrow \exists t \in l \wedge t' \in l', t <_v t' \\ \forall c, c' \in \mathcal{C}^{comp}, c <_h c' &\Leftrightarrow \exists t \in c \wedge t' \in c', t <_h t' \end{aligned}$$

Cet ordre s'étend alors de manière canonique aux lignes et colonnes par :

$$\begin{aligned} \forall l, l' \in \mathcal{L}, l <_v l' &\Leftrightarrow \exists l^c, l'^c \in \mathcal{L}^{comp}, l \subseteq l^c \wedge l' \subseteq l'^c \wedge l^c <_v l'^c \\ \forall c, c' \in \mathcal{C}, c <_h c' &\Leftrightarrow \exists c^c, c'^c \in \mathcal{C}^{comp}, c \subseteq c^c \wedge c' \subseteq c'^c \wedge c^c <_h c'^c \end{aligned}$$

Nous étendons alors le motif $MLC = (TL \cup TC, I)$ en un motif de lignes/colonnes ordonnées $MLC^O = (TL \cup TC, I, O)$ avec $O \subseteq (TL \times TL) \cup (TC \times TC)$, un ensemble de couples décrivant les relations d'ordres devant être respectées par les lignes et colonnes entre elles. Nous pouvons alors ajouter la contrainte d'ordre suivante à notre modèle :

$$\forall u_{lc}, u_{lc'} \in \mathcal{X}, (lc, lc') \in O \Rightarrow (u_{lc} <_v u_{lc'} \vee u_{lc} <_h u_{lc'})$$

3.3 Algorithme de filtrage

L'un de nos objectifs est de mesurer l'impact d'un algorithme de filtrage spécifique sur la recherche de motifs en fonction de nos différents modèles. Nous nous appuyons sur la version simple de l'algorithme *Iterative Labeling Filtering* (ILF) [27]. Cet algorithme de filtrage repose globalement sur un étiquetage des noeuds des graphes en fonction de relations avec les autres noeuds. Comme la plupart des algorithmes de filtrage le processus est itératif jusqu'à l'obtention d'un point fixe.

Nous allons préciser cet algorithme dans le cadre de la recherche d'un motif de positions. Par conséquent, on se donne en entrée un graphe $GP = (\mathcal{T}, \mathcal{A}, \mathcal{E})$, un motif associé $MP = (T, A, E)$ et le CSP associé $(\mathcal{X}, D, \mathcal{C})_P$ afin de pouvoir poser une contrainte globale d'isomorphisme de sous-graphes $SIP(GP, MP, \mathcal{X})$ telle que définie dans [27].

Pour étiqueter les sommets du graphe $GP = (\mathcal{T}, \mathcal{A}, \mathcal{E})$ on définit une fonction totale d'étiquetage (qui n'a pas de lien avec $\mathcal{E}$) $\alpha : \mathcal{T} \cup T \rightarrow \mathbb{L}$, où $\mathbb{L}$ est un ensemble de labels.

GP et MP étant des graphes orientés, on définit classiquement pour tout sommet $t \in T \cup \mathcal{T}$, $deg^+(t)$ et $deg^-(t)$ son degré sortant et entrant. A l'aide de ces degrés, nous construisons alors un ordre partiel² $\preceq \subseteq \mathbb{L} \times \mathbb{L}$ défini ainsi pour $l = \alpha(t)$ et $l' = \alpha(t')$:

$$l \preceq l' \Leftrightarrow deg^-(t) \leq deg^-(t') \wedge deg^+(t) \leq deg^+(t')$$

2. D'autres mesures peuvent être utilisées pour définir cet ordre partiel. Toutefois, après plusieurs expérimentations basées sur les cycles et les distances, nous conservons le degré.

Cet étiquetage permettra ainsi de déterminer les couples d'associations compatibles entre les sommets du graphe motif et ceux du graphe de positions cible.

$$CC = \{(t, t') \in T \times \mathcal{T} \mid \alpha(t) \preceq \alpha(t')\}$$

Il est ainsi possible de filtrer les domaines des variables de $\mathcal{X}$ à l'aide de CC , en supprimant du domaine $D(x_t)$ les valeurs $t' \in \mathcal{T}$ telles que $(t, t') \notin CC$. Dans l'algorithme ILF, cet étiquetage est ensuite itéré sur les voisins des sommets de $T \cup \mathcal{T}$ au moyen d'un ordre multi-ensemble et ce jusqu'à obtention d'un point fixe dans une séquence classique de filtrages successifs des domaines. Nous avons donc repris³ l'algorithme ILF [27] en y introduisant le fait que pour GP , le graphe est dirigé et pour GLC , le graphe est biparti, comme nous le verrons dans les expérimentations de la section suivante.

4 Analyse expérimentale

L'objectif de cette section est de comparer les deux modèles complémentaires proposés en étudiant la recherche d'un motif similaire dans les deux représentations. Nous souhaitons également mesurer l'impact de l'algorithme de filtrage ILF décrit plus haut.

4.1 Configuration des expérimentations

Nous proposons un générateur d'instances consistant à créer deux relations d'alignement à partir d'une matrice binaire M de dimension 10×10 . On définit alors nos tokens comme étant les valeurs non nulles de cette matrice et nos deux relations $<_h$ et $<_v$ en fonction des positions des tokens sur les lignes et colonnes de M . Un paramètre N correspondra au nombre de tokens présents sur les 100 positions possibles. Les graphes de positions GP^{red} et graphes lignes/colonnes GLC correspondant à l'espace de positionnement induit par M seront donc les graphes cibles pour nos expérimentations, au sein desquels nous chercherons différents motifs. Les paramètres des deux exemples choisis pour notre illustration expérimentale sont précisés dans la table 1. Nous avons choisi ici de restreindre cette expérimentation illustrative à des graphes de position réduits (i.e., sans considérer la transitivité des relations). Cela signifie que les motifs recherchés ne peuvent coïncider qu'avec des sous-graphes cibles ne faisant pas intervenir cette transitivité. Dans ce cas, nous pourrions utiliser pour GP^{red} un algorithme simple de recherche de motif en lieu et place de la PPC. Toutefois, notre objectif n'est pas ici la recherche de l'efficacité de la résolution mais l'utilisation des différents modèles de représentation dans un cadre unifié.

Nous nous limiterons ici à la présentation de deux matrices, avec des densités de tokens variées. Toutefois, des expériences similaires sur d'autres instances du même type révèlent des caractéristiques identiques.

Nous proposons alors de rechercher les motifs suivants :

3. Pour des raisons d'espace, nous ne détaillons pas complètement cet algorithme ici et nous renvoyons le lecteur vers l'article original pour plus de précisions.

Cible (GP/GLC)	# sommets	# arcs/arêtes
GP1	50	80
GLC1	311	3346
GP2	40	60
GLC2	210	1386

TABLE 1 – Paramètres des différents graphes cibles .

- carré : motif de taille 2×2 où tous les tokens sont présents et contenant donc deux lignes et deux colonnes (illustré sur la figure 5).
- pan : motif identique au motif carré avec un token supplémentaire sur la première colonne, constituant ainsi une troisième ligne réduite à un seul token. La figure 6 illustre ce motif pan présenté comme un motif de positions et comme un motif lignes/colonnes correspondant.
- gap : motif de dimension 3×2 comportant un token manquant situé à l'intersection de la deuxième ligne et de la première colonne.
- net : un motif de type carré de dimension 3×3 .

Nous avons sélectionné ces motifs car ils nous permettent d'exhiber différents cas pour les ensembles de solution calculés, notamment absence de solutions, en fonction des paramètres du graphe cible. Ils sont par ailleurs liés aux problématiques rencontrés lors des travaux sur l'extraction de tableaux dans des documents, d'où leurs formes assez spécifiques.

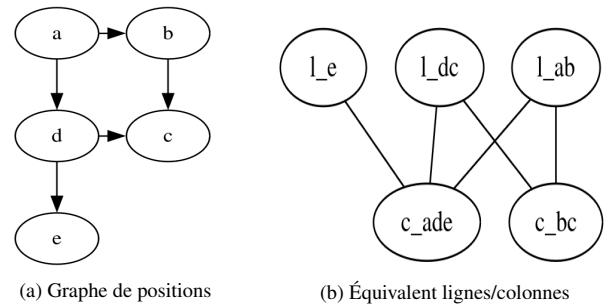


FIGURE 6 – Motif de positions pan 6a et motif lignes/colonnes induit 6b. Les labels des sommets correspondant aux lignes et colonnes utilisent la même convention que précédemment (type(l/c)) suivi du nom des sommets du motif qui sont dans la ligne/colonne correspondante.

Pour évaluer l'efficacité du solveur dans la résolution du problème, nous utilisons les mesures classiques suivantes [2] :

- Sollicitations effectives (effs) : le nombre de sollicitations d'un propagateur ayant eu un effet tangible sur la recherche.
- Affectations échouées (fails) : le nombre de fois où une affectation a été suivie d'un filtrage conduisant immédiatement à un conflit.

- Décisions erronées (Wrgrs) : cette mesure quantifie le nombre de choix incorrects effectués par le solveur. Notamment, cette mesure prend la valeur 0 si le solveur explore l'espace de recherche sans commettre d'erreur.

Toutes les expérimentations ont été menées en utilisant PyCSP3 [12] et le solveur ACE [11] dans une configuration de base.

4.2 Résultats

Motifs de positions : La table 2 correspond aux résultats obtenus pour la recherche de motifs sur les graphes de positions *GP1* et *GP2* avec nos quatre motifs. La colonne **ILF** indique l'utilisation (+) ou non (-) du filtrage ILF lors de la résolution. La colonne **Sol** correspond au nombre de solutions trouvées. Remarquons que les solutions trouvées ici correspondent bien directement aux motifs attendus (ce qui ne sera pas le cas pour les motifs lignes/colonnes). L'indication domaine vide (ILF) correspond à des situations où le filtrage initial permet de réduire directement certains domaines à vide et donc de montrer l'absence de solution.

Cible	Motif	ILF	Effs	Fails	Wrgrs	Sol
GP1	pan	+	3227	12	28	5
		-	31282	84	85	5
	carré	+	3606	11	33	8
		-	22541	81	82	8
	net	+	domaine vide (ILF)			0
		-	26809	42	43	0
	gap	+	6097	61	62	2
		-	26188	69	70	2
GP2	pan	+	344	1	1	5
		-	5003	12	38	5
	carré	+	527	5	7	5
		-	4020	12	38	5
	net	+	domaine vide (ILF)			0
		-	6329	32	33	0
	gap	+	1470	19	20	0
		-	3328	32	33	0

TABLE 2 – Tableau des résultats sur les graphes de positions

Motifs de lignes/colonnes : La table 3 correspond aux résultats obtenus pour la recherche de motifs sur les graphes de lignes/colonnes *GLC1* et *GLC2* avec nos quatre motifs (dans leurs versions de motifs de lignes/colonnes). Les colonnes **ILF** et **Sol** ont naturellement les mêmes significations que précédemment. La colonne **Typage** correspond à l'utilisation (+) ou non (-), déjà évoquée, du type des sommets (ligne ou colonne) pour filtrer les domaines (spécificité du problème d'isomorphisme dans le cas d'un graphe biparti). La colonne **Ordre** correspond à l'ajout (+) ou non (-) de la contrainte d'ordre sur les lignes/colonnes (expliquée dans la sous-section 2.3). Nous observons tout naturellement que l'ajout de **Typage** et **Ordre** réduit le nombre de solutions en éliminant des solutions qui, en réalité, ne correspondent à l'un des motifs de positions calculé. A contra-

rio, en l'absence de **Typage** et **Ordre** certaines configurations ne peuvent être résolues. Dans ce cas, les champs des différentes colonnes affichent la mention "out of memory", indiquant que le solveur a dû faire face à un dépassement de capacité mémoire.

Analyse des résultats : Comme annoncé, ces expérimentations constituent une première analyse computationnelle de notre modélisation. Elles nous permettent toutefois d'observer les phénomènes suivants :

- **Impact d'ILF sur la recherche de motif dans GP :** Dans la table 2, nous observons clairement que l'utilisation du filtrage réduit considérablement la taille de l'arbre de recherche. Il est intéressant de noter que pour le motif net (pour lequel il n'y a pas de solution), ILF suffit à générer des domaines vides. En outre, nous avons observé que la prise en compte de l'orientation des arcs améliore la capacité de filtrage de l'algorithme.
- **Impact d'ILF sur la recherche de motif dans GLC :** Dans le cas des motifs lignes/colonnes, l'introduction des spécificités du graphe biparti dans le modèle (type des sommets et ordre comme mentionné dans la section 3.2) permet de réduire les domaines, ce qui rend l'utilisation d'ILF inutile.
- **Comparaisons des solutions obtenues sur GP et GLC :** Nous pouvons tout d'abord vérifier, que lorsqu'il n'y a pas de solutions, les deux modèles coïncident bien. Les différences notables entre les nombres de solutions obtenues pour un motif lignes/colonnes par rapport à un motif de positions correspondant est d'une part due à la notion d'ordre qui a déjà été illustrée sur notre exemple dans la section 2.3, mais également car le motif lignes/colonnes ne prend pas en compte la cardinalité des lignes et colonnes recherchées. Plus précisément certaines solutions obtenues correspondent à des lignes colonnes contenant plus (ou moins) de tokens que le motif de positions initial. La recherche d'un motif lignes/colonnes peut être vue comme une étape intermédiaire. Les solutions obtenues doivent ensuite être restreintes via une contrainte assurant une cohérence entre le nombre de tokens présents dans les lignes et colonnes du motif de positions et la cardinalité implicite des lignes et colonnes du motif lignes/colonnes. L'équivalence existentielle mise en avant par la propriété 2 ne nous garantit en effet pas directement, comme c'est le cas pour l'ordre des lignes et colonnes, une correspondance stricte des solutions. L'ajout d'une contrainte de cardinalité nous a permis de valider cette extension et constitue une piste de développement future pour l'amélioration du filtrage. Remarquons par ailleurs que, en utilisant les contraintes de typage et d'ordre sur les lignes/colonnes, si le nombre de solutions est important, l'arbre de résolution est toutefois plus compact que pour la recherche du motif de positions.

Cible	Motif	ILF	Typage	Ordre	Effs	Fails	Wrgrs	Solutions
GLC1	pan	+	-	-	out of memory			
			+	-	2573	42	58	4236
			+	+	1408	35	46	1972
		-	-	-	out of memory			
			+	-	out of memory			
			+	+	1408	35	46	1972
	carré	+	-	-	253173	148	168	2525552
			+	-	128873	107	128	1262776
			+	+	746	10	13	2471
		-	-	-	253173	148	168	2525552
			+	-	128873	107	128	1262776
			+	+	746	10	13	2471
	net	+	-	-	1300	154	155	0
			+	-	1	0	0	0
			+	+	1	0	0	0
		-	-	-	1325	155	156	0
			+	-	1	0	0	0
			+	+	1	0	0	0
	gap	+	-	-	out of memory			
			+	-	2155	41	51	5068
			+	+	811	13	18	1675
		-	-	-	out of memory			
			+	-	out of memory			
			+	+	811	13	18	1675
GLC2	pan	+	-	-	out of memory			
			+	-	581834	3205	3787	3377800
			+	+	451	34	38	186
		-	-	-	out of memory			
			+	-	585407	3347	3959	3377800
			+	+	453	34	38	186
	carré	+	-	-	67135	128	161	345024
			+	-	33302	62	67	172512
			+	+	361	22	23	395
		-	-	-	67135	128	161	345024
			+	-	33302	62	67	172512
			+	+	361	22	23	395
	net	+	-	-	875	100	101	0
			+	-	1	0	0	0
			+	+	1	0	0	0
		-	-	-	944	108	109	0
			+	-	1	0	0	0
			+	+	1	0	0	0
	gap	+	-	-	out of memory			
			+	-	582308	2378	3029	3377800
			+	+	252	7	7	192
		-	-	-	out of memory			
			+	-	568372	3093	3656	3377800
			+	+	252	7	7	192

TABLE 3 – Tableau des résultats sur les graphes lignes/colonnes

5 Perspectives

Ce travail constitue une première étape dans la définition d'un formalisme de description d'espaces de positionnements d'objets via des relations simples d'alignements qui peut trouver diverses applications potentielles. La définition de plusieurs modèles nous a permis de présenter différentes approches possibles pour rechercher des motifs au sein de ces espaces de positionnement. En mettant en avant des correspondances entre ces représentations, nous avons également observé qu'elles ne visaient pas forcément, dans leurs versions initiales, les mêmes ensembles de solutions (certaines pouvant être vues comme des relaxations). Nous avons pu toutefois préciser les contraintes additionnelles nécessaires pour extraire précisément les structures recherchées.

Les perspectives sont nombreuses, tant du point de vue de l'extension du formalisme pour affiner les équivalences et les complémentarités entre les différents modèles de graphes, que du point de vue de la résolution du problème en insérant des contraintes telles que la cardinalité au niveau du filtrage et en définissant, par exemple, une approche de résolution multi-niveaux, faisant appel aux différents modèles de manière entrelacée.

Remerciements : Nous remercions les relecteurs pour leurs remarques et suggestions pertinentes qui ont contribué à identifier des pistes de recherche futures.

Références

- [1] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11) :832–843, November 1983.
- [2] Christian Bessiere, Bruno Zanuttini, and Cèsar Fernández. Measuring Search Trees. In Hnich B. and Walsh T., editors, *Workshop on Modelling and Solving Problems with Constraints - ECAI'2004*, pages 31–40, Valencia, Spain, 2004. Hnich B. Workshop held in conjunction with the 16th European Conference on Artificial Intelligence (ECAI 2004).
- [3] Sanket Biswas, Pau Riba, Josep Lladós, and Uma-pada Pal. Graph-based deep generative modelling for document layout generation. *CoRR*, abs/2107.04357, 2021.
- [4] Anthony G. Cohn, Brandon Bennett, John Gooday, and Nicholas Mark Gotts. Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*, 1(3) :275–316, 1997.
- [5] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10) :1367–1372, 2004.
- [6] M. R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [7] Andrea Gemelli, Emanuele Vivoli, and Simone Marini. Graph neural networks and representation em-

- bedding for table extraction in PDF documents. In *26th International Conference on Pattern Recognition, ICPR 2022, Montreal, QC, Canada, August 21-25, 2022*, pages 1719–1726. IEEE, 2022.
- [8] Gael Glorian, Jean-Marie Lagniez, Valentin Montmireil, and Michael Sioutis. Une approche SAT incrémentale pour raisonner efficacement sur les réseaux de contraintes qualitatives. In *JFPC 2019 - Actes des 15es Journées Francophones de Programmation par Contraintes*, Albi, France, June 2019.
- [9] Bo Huang, Thomas Cova, Ming-Hsiang Tsou, Georg Bareth, Chunqiao Song, Yan Song, Kai Cao, and Elisabete Silva. *Comprehensive Geographic Information Systems*. 08 2017.
- [10] Lars Kotthoff, Ciaran McCreesh, and Christine Solnon. Portfolios of subgraph isomorphism algorithms. In Paola Festa, Meinolf Sellmann, and Joaquin Vanschoren, editors, *Learning and Intelligent Optimization - 10th International Conference, LION 10, Ischia, Italy, May 29 - June 1, 2016, Revised Selected Papers*, volume 10079 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 2016.
- [11] Christophe Lecoutre. Ace, a generic constraint solver, 2024.
- [12] Christophe Lecoutre and Nicolas Szczepanski. Pycsp3 : Modeling combinatorial constrained problems in python, 2024.
- [13] Alan K. Mackworth. Interpreting pictures of polyhedral scenes. In Nils J. Nilsson, editor, *Proceedings of the 3rd International Joint Conference on Artificial Intelligence, Stanford, CA, USA, August 20-23, 1973*, pages 556–563. William Kaufmann, 1973.
- [14] Ciaran McCreesh, Patrick Prosser, and James Trimble. The glasgow subgraph solver : Using constraint programming to tackle hard subgraph isomorphism problem variants. In Fabio Gadducci and Timo Kehrer, editors, *Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings*, volume 12150 of *Lecture Notes in Computer Science*, pages 316–324. Springer, 2020.
- [15] Ugo Montanari. Networks of constraints : Fundamental properties and applications to picture processing. *Inf. Sci.*, 7 :95–132, 1974.
- [16] Charalampos Nikolaou and Manolis Koubarakis. Fast consistency checking of very large real-world rcc-8 constraint networks using graph partitioning. *Proceedings of the National Conference on Artificial Intelligence*, 4 :2724–2730, 06 2014.
- [17] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In Barbara Hayes-Roth and Richard E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1*, pages 362–367. AAAI Press / The MIT Press, 1994.
- [18] Jochen Renz and Bernhard Nebel. On the complexity of qualitative spatial reasoning : A maximal tractable fragment of the region connection calculus. *Artif. Intell.*, 108(1-2) :69–123, 1999.
- [19] Pau Riba, Anjan Dutta, Lutz Goldmann, Alicia Fornés, Oriol Ramos Terrades, and Josep Lladós. Table detection in invoice documents by graph neural networks. In *2019 ICDAR*. IEEE, 2019.
- [20] Mahmoud Salaheldin Kasem, Abdelrahman Abdallah, Alexander Berendeyev, Ebrahim Elkady, Mohamed Mahmoud, Mahmoud Abdalla, Mohamed Hamada, Sebastiano Vascon, Daniyar Nurseitov, and Islam Taj-Eddin. Deep learning for table detection and structure recognition : A survey. *ACM Comput. Surv.*, 56(12), October 2024.
- [21] Thomas Saout, Frédéric Lardeux, and Frédéric Saubion. A two-stage approach for tables extraction in invoices. In *35th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2023, Atlanta, GA, USA, November 6-8, 2023*, pages 10–15. IEEE, 2023.
- [22] Thomas Saout, Frédéric Lardeux, and Frédéric Saubion. An overview of data extraction from invoices. *IEEE Access*, 12 :19872–19886, 2024.
- [23] Christine Solnon. Alldifferent-based filtering for subgraph isomorphism. *Artif. Intell.*, 174(12-13) :850–864, 2010.
- [24] Julian R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1) :31–42, 1976.
- [25] Yang Wang, Huilin Peng, Yiwei Xiong, and Haitao Song. Spatial relationship recognition via heterogeneous representation : A review. *Neurocomputing*, 533 :116–140, 2023.
- [26] Yiyuan Wang, Chenghou Jin, and Shaowei Cai. Pathlad+ : Towards effective exact methods for subgraph isomorphism problem. *Artif. Intell.*, 337 :104219, 2024.
- [27] Stéphane Zampelli, Yves Deville, and Christine Solnon. Solving subgraph isomorphism problems with constraint programming. *Constraints*, 15(3) :327–353, 2010.

Session 7a, mercredi 2 juillet, 10h30-11h25 : Contraintes

Amélioration de la contrainte globale de cardinalité avec coûts par l'introduction de points de repère *

Margaux Schmied¹, Jean-Charles Régim¹

¹ Université Côte d'Azur, CNRS, I3S, France

margaux.schmied@icloud.com, jcregin@gmail.com

Résumé

Le succès de la programmation par contraintes repose en partie sur l'efficacité des contraintes globales et de leurs algorithmes de filtrage. Nous étudions ici la contrainte globale de cardinalité avec coûts, qui généralise la contrainte de différence en limitant le nombre d'affectations par valeur et en imposant une contrainte sur le coût des affectations. L'algorithme classique de filtrage repose sur le calcul systématique de plus courts chemins, ce qui le rend coûteux en pratique. Nous proposons une approche plus rapide, fondée sur des bornes supérieures obtenues via des points de repère (landmarks en anglais), réduisant ainsi significativement les calculs explicites nécessaires. Les expériences montrent que, dans le meilleur des cas, elle peut être en moyenne 57 fois plus rapide.

Mots-clés

Globale contrainte, Algorithme de filtrage, Contrainte globale de cardinalité avec coûts, Arc consistance

Abstract

The success of constraint programming relies partly on the efficiency of global constraints and their filtering algorithms. We study in this article the global cardinality constraint with costs, which generalizes the difference constraint by limiting the number of assignments per value and imposing a constraint on the cost of assignments. The classical filtering algorithm is based on the systematic computation of shortest paths, which makes it costly in practice. We propose a faster approach, based on upper bounds obtained via landmarks, thus significantly reducing the explicit computations required. Experiments show that, in the best case, it is on average up to 57 times faster.

Keywords

Global constraint, Filtering algorithm, Cardinality constraints with costs, Arc consistency

1 Introduction

En Programmation par Contraintes (PPC), un problème est défini par des variables et des contraintes. Chaque variable possède un domaine qui représente l'ensemble de ses valeurs possibles, tandis qu'une contrainte exprime une re-

lation à satisfaire entre un groupe de variables. La PPC s'appuie sur des méthodes de résolution adaptées à chaque contrainte, appelées algorithmes de filtrage.

Le succès de la PPC repose en partie sur l'efficacité de ces algorithmes. Ils éliminent des valeurs dans les domaines des variables lorsque celles-ci ne peuvent pas appartenir à une solution satisfaisant la contrainte. En d'autres termes, ils garantissent la consistance des contraintes en simplifiant le problème à résoudre.

Nous nous intéressons à la contrainte globale de cardinalité avec coûts (costGCC) [6]. Cette contrainte généralise la contrainte de différence. Elle s'applique à un ensemble de variables $X = x_1, \dots, x_p$ avec des valeurs $V = v_1, \dots, v_d$. Chaque valeur $v_i \in V$ doit être affectée entre $[l_i, u_i]$ fois. Un coût est associé à chaque affectation, et le coût total des affectations doit être inférieur à H . Cette contrainte est utilisée pour le routage ou l'ordonnancement, où les coûts peuvent modéliser des préférences, des poids ou des durées. L'algorithme de filtrage associé à la contrainte costGCC [6] peut être qualifié de répétitif. Il commence par calculer un flot maximum à coût minimum (minCostMaxFlow) afin de vérifier si la contrainte est consistante, c'est-à-dire si une solution existe. Puis, il élimine les valeurs inconsistantes avec la contrainte, autrement dit qui ne peuvent pas appartenir à une solution. Plus précisément, il examine chaque valeur de chaque variable afin de déterminer s'il existe un chemin dans le graphe résiduel dont la valeur est inférieure à H .

Nous présentons une nouvelle approche visant à minimiser l'aspect répétitif de l'algorithme de filtrage. Cette approche repose sur plusieurs observations :

- **Flot suffisant** : Un flot à coût minimal n'est pas nécessaire pour chaque affectation, une simple vérification d'un coût inférieur à H suffit.
- **Approximation des coûts** : Des bornes supérieures approximatives sur les coûts des plus courts chemins sont suffisantes.
- **Points de repère** : L'utilisation de nœuds particuliers, appelés points de repère, s'avère efficace pour accélérer le calcul des plus courts chemins dans de grands graphes (contenant des millions de nœuds) [2]. Soient x et y deux nœuds d'un graphe et p un point de repère, alors $d(x, p) + d(p, y) \geq d(x, y)$ est toujours vérifiée. Cela permet d'obte-

*Cet article se base sur des résultats publiés à CP 2024[7].

nir immédiatement une borne supérieure de $d(x, y)$ entre chaque paire de nœuds x et y .

- **Impact des suppressions** : En début de recherche, le filtrage n'élimine souvent aucune valeur. Cela suggère que la marge entre H et le coût du flot minimal est généralement importante, justifiant l'usage d'approximation.

Pour améliorer l'algorithme proposé par Régim [6], nous proposons une phase de prétraitement pour limiter les calculs explicites des plus courts chemins. Notre méthode identifie des points de repère dans le graphe, permettant d'estimer des bornes supérieures sur les coûts des chemins réduisant les calculs inutiles. Ces points sont choisis selon la structure du graphe (centre (C), périphérie (O), centre et périphérie (C & O), degré des nœuds (Deg) et aléatoire (R)) : deux calculs de plus courts chemins suffisent par repère. Nous introduisons une propriété pour détecter rapidement l'arc consistant d'une contrainte costGCC, réduisant davantage les opérations superflues.

Le Tableau 1 présente les ratios moyens d'accélération (Régim/Landmarks) obtenus avec notre méthode sur quatre ensembles de données : 1) Problème du voyageur de commerce (TSP) [1], 2) problème de stockage avec coûts (StockingCost) [3], 3) problème d'ordonnancement de tâches (FJSSP) [5, 9] et 4) problème d'assignation d'enfants à des activités (CHILD) [8]. La valeur de seuil H pour le TSP provient de l'heuristique de Lin-Kernighan [4]. Pour les autres instances, H correspond à la plus petite valeur garantissant l'existence d'une solution.

Ces résultats montrent que notre approche est toujours au moins aussi rapide que la méthode de Régim et qu'elle peut être en moyenne jusqu'à 57 fois plus rapide dans les meilleurs cas.

	C	O	C & O	Deg	R
TSP (≤ 100 cities)	1.2	1.2	1.1	1.3	1.6
TSP (> 100 & < 250 cities)	2.6	2.5	2.5	2.7	2.5
TSP (≥ 250 cities)	43.5	44.1	44	56.9	45.8
StockingCost (H)	1.2	1	1	1	0.9
StockingCost ($H \times 2$)	15.7	16.5	16.9	16	16.4
FJSSP (H)	0.8	1.7	0.75	1	0.8
FJSSP ($H \times 2$)	1	1.3	1.3	1.3	1.3
CHILD (H)	0.9	1.2	1	0.8	1
CHILD ($H \times 2$)	8.3	9	8	9.7	9.7

TABLE 1 – Ratio d'accélération (Régim/landmarks) du calcul de l'arc consistant d'une contrainte costGCC avec 4 repères.

2 Conclusion

Nous présentons une implémentation optimisée de l'algorithme d'arc consistant pour la contrainte costGCC, essentielle dans de nombreux problèmes industriels. L'algorithme classique, basé sur le calcul exact des plus courts chemins, est souvent trop lent en pratique. Nous proposons une approche reposant sur des bornes supérieures issues d'inégalités triangulaires et de points de repère, réduisant ainsi le nombre de calculs nécessaires. Les gains sont d'au-

tant plus significatifs que plus le graphe est grand plus les coûts de calcul sont important. De plus, nous introduisons une condition suffisante, simple et rapide à évaluer, permettant de vérifier efficacement si une contrainte costGCC est arc consistante.

Remerciements

Ce travail a été soutenu par le gouvernement français, à travers le projet 3IA Côte d'Azur Investissements d'Avenir géré par l'Agence Nationale de la Recherche (ANR) avec le numéro de référence ANR-19-P3IA-0002.

Références

- [1] R. Gerhard. TSPLIB—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4) :376–384, 1991.
- [2] A. V. Goldberg and C. Harrelson. Computing the shortest path : A search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, page 156–165, USA, 2005. Society for Industrial and Applied Mathematics.
- [3] V. R. Houndji, P. Schaus, and L. Wolsey. The item dependent stockingcost constraint. *Constraints*, 24(2) :183–209, April 2019.
- [4] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.*, 21 :498–516, 1973.
- [5] M. Pelleau, A. Miné, C. Truchet, and F. Benhamou. A constraint solver based on abstract domains. In *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, pages 434–454, 2013.
- [6] J.-C. Régim. Cost-based arc consistency for global cardinality constraints. *Constraints*, 7(3/4) :387–405, jul 2002.
- [7] M. Schmied and J.-C. Régim. Efficient Implementation of the Global Cardinality Constraint with Costs. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307, pages 27 :1–27 :18, 2024.
- [8] S. Varone and C. Beffa. Dataset on a problem of assigning activities to children, with various optimization constraints. *Data in Brief*, 2019.
- [9] T. Weise. jsspinstancesandresults : Results, data, and instances of the job shop scheduling problem, 2019–2020. A GitHub repository with the common benchmark instances for the Job Shop Scheduling Problem as well as results from the literature, both in form of CSV files as well as R program code to access them.

Activation de règles CHR pour la composition et l'orchestration de solveurs par contraintes

Victor Allard, Vincent Barichard, Eric Monfroy, Jean-Michel Richer

LERIA, Université d'Angers

{victor.allard,vincent.barichard,eric.monfroy,jean-michel.richer}@univ-angers.fr

Résumé

CHR est un langage à base de règles et de contraintes pour modéliser et résoudre des problèmes combinatoires. Contrairement aux solveurs dédiés à un langage de modélisation tel que CSP, l'utilisateur écrit les règles qui simplifieront le problème jusqu'à sa résolution. Nous étudions ici différents algorithmes de résolution (méthodes de filtrage et d'énumération), ainsi que leur composition par activation et désactivation des composants. Il est alors possible de définir des stratégies de résolution flexibles et adaptatives, évoluant durant la résolution. Ce travail préliminaire en CHR++, ouvre la voie à la conception de solveurs performants pouvant ajuster leur stratégie dynamiquement durant la résolution.

Mots-clés

stratégie, composition de solveurs, Constraint handling rules, Constraint (logic) programming, CSP, CHR++

Abstract

CHR is a rule- and constraint-based language for modeling and solving combinatorial problems. Unlike solvers dedicated to a modeling language such as CSP, the user writes the rules that simplify the problem until it is solved. Here, we examine various solving algorithms (filtering and enumeration methods), as well as their composition through the activation and deactivation of components. It is then possible to define flexible and adaptive solving strategies that evolve during resolution. This preliminary work in CHR++ paves the way for the design of high-performance solvers capable of dynamically adjusting their strategy during resolution.

Keywords

strategy, solvers composition, Constraint handling rules, Constraint logic programming systems, CSP, CHR++

1 Introduction

La résolution de problèmes combinatoires sous contraintes est un domaine largement étudié, offrant de nombreuses applications concrètes, notamment en logistique [11], planification [5], allocation de ressources [13], ... Pour traiter ces problèmes, un large éventail de méthodes a été développé, qu'elles soient

exactes ou approchées.

Dans le cadre des approches exactes, plusieurs méthodes ont été développées parmi lesquelles : méthode analytique de réécriture (méthode de Gauss), méthode de calcul numérique (Simplex), méthode d'exploration arborescente (élagage alpha-bêta [8], CSP [10], ...). Certaines de ces approches, notamment les CSPs, reposent sur un langage de modélisation de haut niveau associé à un solveur chargé de résoudre le problème. Ces solveurs sont construits autour d'un ensemble de composants, chacun étant dédié à un aspect clé de la méthode. Le formalisme CSP, en particulier, permet de modéliser précisément les propriétés du problème avant de les soumettre à un solveur. Celui-ci est généralement conçu avec une architecture statique, où les composants et algorithmes sont sélectionnés dès l'implémentation, en fonction des performances moyennes observées. Dans ce travail préliminaire, nous nous proposons de définir un langage de stratégie permettant d'organiser dynamiquement les composants d'un solveur. Nous concentrerons d'abord notre étude sur les composants des solveurs CSP afin d'analyser les opérateurs existants, d'enrichir leur palette par l'ajout de nouveaux opérateurs et d'élaborer des stratégies simples. Nous présentons les CSPs ainsi que les différents composants que nous étudierons en section 2. La section 3 présente CHR, notre langage de représentation et d'implémentation. La section 4 est consacrée à l'étude expérimentale des différents composants et l'analyse des premiers résultats. Nous concluons en section 5, en proposant des perspectives d'amélioration.

2 CSP

Un problème modélisé par un CSP est fourni en entrée d'un solveur en charge de le résoudre.

Définition 1 (CSP) *Un CSP (Constraint Satisfaction Problem) est un problème modélisé sous forme de contraintes. X est l'ensemble des variables $\{X_1, \dots, X_n\}$ qui prennent leurs valeurs dans un ensemble de domaines $D = \{D_1, \dots, D_n\}$, avec $X_i \in D_i \forall i \in [1, n]$. Enfin, $C = \{C_1, \dots, C_m\}$ est l'ensemble des contraintes exprimées sur les variables. Résoudre un CSP revient à trouver une assignation, pour chaque*

variable de X , qui satisfait toutes les contraintes de C .

Les solveurs CSP, tels que Gecode [12], offrent la possibilité de paramétrer la recherche de solutions en sélectionnant un ensemble d'heuristiques pour guider les processus de branchement et de filtrage. Gecode permet notamment de configurer les stratégies de branchement, de contrôler les mécanismes de redémarrage de la recherche et de sélectionner la puissance de certains opérateurs de filtrage¹.

Cependant, ces paramètres restent statiques, car les choix de configuration sont fixés avant le démarrage de la recherche et ne peuvent pas être adaptés dynamiquement en cours d'exécution. Pour étudier, réorganiser et combiner les différents composants d'un solveur, il est essentiel de les implémenter dans un cadre flexible qui permette leur réutilisation sans nécessiter un redéveloppement complet. Pour cela, l'utilisation d'un langage de haut niveau spécialement adapté à la conception de solveurs est indispensable.

3 CHR

CHR (*Constraint Handling Rules*) est un langage à base de règles, offrant un haut niveau d'abstraction. Il permet de décrire les propriétés d'un problème, mais aussi d'étendre ses capacités en ajoutant de nouvelles règles. À la fois langage de modélisation et de programmation, il constitue un cadre très adapté au développement de solveurs. CHR est utilisé pour implémenter et étudier les différents composants d'un solveur CSP. Ces composants sont implémentés sous forme de règles, pouvant être activées ou désactivées dynamiquement. Cette approche permet de composer et d'orchestrer efficacement les différents éléments du solveur.

3.1 Présentation

CHR est un langage de programmation à chaînage avant à base de règles et de contraintes [6], qui nécessite un langage hôte pour être implémenté. CHR permet de déclarer des règles qui réécrivent des contraintes (des relations entre variables logiques), sous certaines conditions, jusqu'à la rencontre d'un critère de terminaison. Nous utilisons ici CHR[∇] [3], une extension de CHR intégrant la disjonction. Cette extension est particulièrement adaptée à la modélisation de problèmes combinatoires et à la conception d'algorithmes de résolution efficaces reposant sur une exploration arborescente.

3.1.1 Règles CHR

Un programme CHR est un ensemble fini de règles. Dès lors qu'une contrainte est activée, le programme recherche les règles qui peuvent être déclenchées. Chaque règle se compose d'une tête, d'une garde optionnelle et d'un corps. La tête est une conjonction de contraintes qui doivent être présentes dans le store de contraintes pour que la règle puisse s'appliquer. La garde est une

condition qui doit être satisfaite pour autoriser l'activation de la règle. Le corps définit une séquence de contraintes qui sont ajoutées au but lors du déclenchement de la règle. Il existe trois types de règles :

- Propagation : $H \Rightarrow G \mid B$
- Simplification : $H \Leftrightarrow G \mid B$
- Simpagation : $H_k \setminus H_r \Leftrightarrow G \mid B$

Où H , H_k et H_r représentent des multiensembles de contraintes, G une garde et B le corps d'une règle. Dans une règle de propagation, les contraintes H utilisées pour le déclenchement sont conservées dans le store de contraintes. Dans une règle de simplification, les contraintes H sont supprimées du store de contraintes au moment de son activation. Lors de l'activation d'une règle de simpagation, le multiensemble de contraintes H_k est conservé, tandis que le multiensemble de contraintes H_r est supprimé du store de contraintes.

3.1.2 Exemple de programme CHR

Le programme CHR présenté à la figure 1 permet de calculer le plus grand commun diviseur (PGCD) de deux nombres. Initialement, deux contraintes représentant les conditions de départ du problème (les deux nombres dont on souhaite déterminer le PGCD) sont ajoutées. Ensuite, la règle 2 réécrit les contraintes de la tête en d'autres plus simples jusqu'à ce que la règle 1 soit activée, mettant ainsi fin au calcul.

$$\begin{aligned} gcd(0) & \Leftrightarrow \top & (1) \\ gcd(N) \setminus gcd(M) & \Leftrightarrow N \leq M \mid gcd(M - N) & (2) \end{aligned}$$

FIGURE 1 – Programme CHR du PGCD

3.2 Solveur CSP en CHR

Nous proposons d'étudier les composants d'un solveur CSP, tels que les opérateurs de filtrage et la fonction d'énumération. Chacun de ces composants est implémenté sous forme d'un ensemble de règles CHR.

Certaines heuristiques ont démontré leur efficacité comme l'ordonnancement dynamiquement de variables (DVO) [1]. Cependant, il existe plusieurs heuristiques DVO concurrentes et la représentation de ces heuristiques dans des composants interchangeable offre la possibilité de choisir l'heuristique DVO avant la recherche et d'en changer pendant la recherche.

La figure 2 illustre deux règles implémentant des stratégies d'énumération distinctes. La règle `dom` explore successivement toutes les valeurs du domaine d'une variable, tandis que `bin` applique un branchement binaire : elle assigne une valeur à la variable sur une branche et l'exclut sur l'autre. Pour enchaîner les points de choix, les contraintes CHR $n_{var}()$ et $n_{val}()$ sont postées. Bien que non détaillées ici, ces contraintes permettent d'imbriquer récursivement les points de choix en ajoutant les contraintes CHR nécessaires pour déclencher l'une des deux règles présentées dans la figure 2. Dans la

1. Voir Chapitres 8, 9 et Partie P : <https://www.gecode.org/doc/6.2.0/MPG.pdf>

règle *dom*, la variable hôte d permet de propager le domaine modifié à chaque opérande de la disjonction et la contrainte $n_{dom}(d, A)$ fixe la variable logique A à la prochaine valeur branchée sur le domaine d .

$$v(X, D_x) \setminus b_{var}(X), b_{val}(V) \Leftrightarrow$$

$$d \leftarrow D_x,$$

$$\bigvee_{y \in d} (n_{dom}(d, A), d \leftarrow d/\{A\}, D_x \leftarrow \{x\}, n_{var}(), n_{val}())$$

(dom)

$$v(X, D_x) \setminus b_{var}(X), b_{val}(V) \Leftrightarrow$$

$$(D_x \leftarrow \{V\}, n_{var}(), n_{val}())$$

$$\vee (D_x \leftarrow D_x/\{V\}, b_{var}(X), n_{val}())$$

(bin)

FIGURE 2 – Énumérations en CHR[∀]

Un solveur CSP sélectionne un type d'énumération à priori, avant le début de la résolution. Pour permettre un changement dynamique de stratégie d'énumération, nous activons ou désactivons certaines règles en cours d'exécution. Cette approche impacte directement le comportement du programme et modifie ainsi le fonctionnement initial du solveur. Dans notre étude expérimentale, nous avons réimplémenté plusieurs composants d'un solveur CSP. Implémentés sous forme de règles CHR, ces composants peuvent être activés ou désactivés dynamiquement selon les besoins.

4 Expérimentations

L'étude expérimentale présentée ici vise à combiner et évaluer différentes versions des composants d'un solveur CSP en les appliquant à un problème bien connu : le problème des N-Reines.

4.1 Le problème des N-Reines.

Le problème des N-Reines est un problème classique de la programmation par contraintes. Il consiste à placer n reines sur un échiquier de taille $n \times n$ de manière à ce qu'aucune d'elles ne puisse en attaquer une autre, que ce soit sur une même ligne, colonne ou diagonale. La figure 3 propose un modèle CSP permettant de calculer une solution à ce problème, si elle existe. La taille de l'échiquier $n \in \mathcal{N}$ est un paramètre du modèle qui permet de générer l'instance à résoudre. La contrainte $all_diff(S)$ est satisfaite si toutes les variables assignées de S prennent des valeurs distinctes.

$$X = \{X_1, \dots, X_n\}$$

$$D = (D_i = 1..n \mid i \in 1..n)$$

$$C = \{all_diff(X),$$

$$all_diff(\{X_i + i \mid i \in 1..n\}),$$

$$all_diff(\{X_i - i \mid i \in 1..n\})\}$$

FIGURE 3 – Modèle CSP des N-Reines

Nous avons traduit ce modèle en CHR, où la contrainte $all_diff()$ a été convertie en un ensemble de règles assurant le test de satisfiabilité, le filtrage de domaine (forward checking [7]) et le maintien de l'arc-consistance.

4.2 Résultats

Ce travail est validé expérimentalement par l'utilisation du solveur CHR++ qui est une implémentation CHR efficace en C++ qui supporte le non-déterminisme *don't know* [3].

Nous avons choisi les composants suivants pour implémenter le solveur CHR : branchement sur une variable $Vr \in \{first, min, deg\}$, branchement sur une valeur $Vl \in \{first\}$, contrainte $Ctr \in \{diff\}$, propagateur $Pr \in \{none, diff\}$, consistance $Cs \in \{none, ac_1\}$, énumération $Lb \in \{dom, bin\}$.

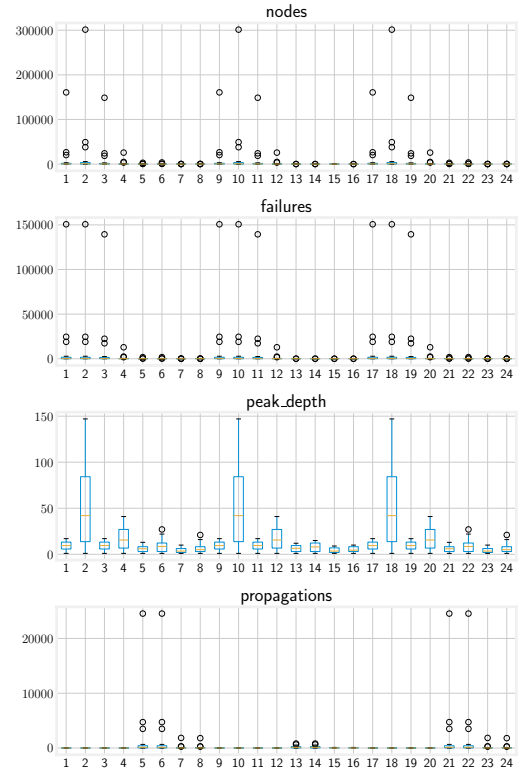


FIGURE 4 – Distribution des indicateurs de recherche sur les N-Reines, N variant de 1 à 16.

Nous avons testé différentes configurations de solveurs CSP en cherchant à déterminer la satisfiabilité des instances $1 \leq N \leq 16$ du modèle des N-Reines. La figure 4 montre la distribution des indicateurs de recherche : nombre de nœuds de l'arbre de recherche (nodes), nombre d'échecs après avoir fixé une variable CSP (failures), profondeur de l'arbre de recherche (peak_depth), nombre de propagations effectuées (propagations). Les 24 compositions différentes ($Vr \times Vl \times Ctr \times Pr \times Cs \times Lb$) ont été testées. L'analyse qui suit se concentre sur la composition numéro 15 de la figure 4, car elle a une des moyennes et des écarts

type la plus basse sur les indicateurs node, failure et peak_depth.

La figure 5 montre l'évolution des indicateurs de recherche pour la composition suivante : $Vr = \min$ qui branche en priorité les variables qui ont le plus petit domaine, $Vl = \text{first}$ qui branche les valeurs dans l'ordre de définition, $Ctr = \text{diff}$ qui représente la contrainte binaire $\neq_2$, $Pr = \text{diff}$ qui représente le propagateur de la contrainte binaire $\neq_2$, $Cs = AC_1$ qui représente une implémentation naïve de l'arc consistance [9], $Lb = \text{dom}$ qui correspond à la règle [dom](#) de la figure 2. On remarque sur cette instance que le nombre de nœuds est proportionnel au nombre d'échecs, ce qui est prévisible, car chaque échec produit de nouveaux points de choix. Les instances $N \in \{5, 7, 9, 16\}$ se démarquent des autres, car elles sont satisfiables et car le maintien de l'arc consistance a permis d'éviter les échecs lors de la recherche. L'étude de ce type d'instance pourrait s'avérer intéressante pour détecter quand l'activation de l'arc consistance est utile.

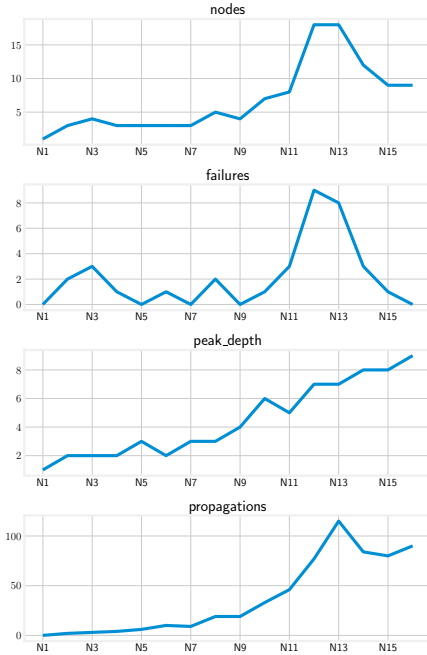


FIGURE 5 – Indicateurs pour la composition 15

5 Conclusion

L'objectif de ce travail est de présenter un système de stratégies au-dessus de CHR qui permet à la manière d'ELAN [4] et de Tom [2], de définir la sémantique à appliquer pour exécuter un ensemble de règles CHR. L'élaboration d'un langage déclaratif d'application et de combinaison d'opérateurs de stratégies est en cours d'élaboration avec pour objectif de l'intégrer dans le solveur CHR++. La prochaine étape de ce travail est donc de trouver des stratégies efficaces qui font varier les composants du solveur CSP pendant la recherche.

Remerciements Nous tenons à remercier l'université d'Angers (UA) et la région Pays de la Loire qui ont financé ce travail.

Références

- [1] Fahiem Bacchus and Paul van Run. Dynamic variable ordering in CSPs. In Ugo Montanari and Francesca Rossi, editors, *Principles and Practice of Constraint Programming — CP '95*, pages 258–275, Berlin, Heidelberg, 1995. Springer.
- [2] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom : Piggybacking rewriting on java. In Franz Baader, editor, *Term Rewriting and Applications*, pages 36–47. Springer Berlin Heidelberg, 2007.
- [3] Vincent Barichard. CHR++ : An efficient CHR system in C++ with don't know non-determinism. *Expert Systems with Applications*, 238 :121810, 2024.
- [4] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Marian Vittek. ELAN : A logical framework based on computational systems. *Electronic Notes in Theoretical Computer Science*, 4 :35–50, 1996. RWLW96, First International Workshop on Rewriting Logic and its Applications.
- [5] Jeremy Frank and Ari Jónsson. Constraint-based attribute and interval planning. *Constraints*, 8(4) :339–364, Oct 2003.
- [6] Thom Frühwirth. *Constraint handling rules*. Cambridge University Press, New York, 2009.
- [7] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *AI*, 14(3) :263–313, 1980.
- [8] Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *AI*, 6(4) :293–326, 1975.
- [9] Alan K. Mackworth. Consistency in networks of relations. *AI*, 8(1) :99–118, 1977.
- [10] Stuart Russell and Peter Norvig. *Artificial Intelligence, Global Edition A Modern Approach*, chapter Chapter 5 : Constraint Satisfaction Problems. Pearson Deutschland, 2021.
- [11] Abdelkader Sbihi and Richard W. Eglese. Combinatorial optimization and Green Logistics. *4OR : A Quarterly Journal of Operations Research*, 5(2) :99–116, June 2007. 18 pages.
- [12] Christian Schulte and Peter J. Stuckey. Speeding up constraint propagation. In Mark Wallace, editor, *Principles and Practice of Constraint Programming — CP 2004*, pages 619–633, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
- [13] G. Tychogiorgos and K.K. Leung. Optimization-based resource allocation in communication networks. *Computer Networks*, 66 :32–45, 2014. Leonard Kleinrock Tribute Issue : A Collection of Papers by his Students.

Réseau de Contrainte Ternaire pour une Propagation Efficace de Bornes sur GPU

Pierre Talbot
Université du Luxembourg
pierre.talbot@uni.lu

Résumé

L'apprentissage automatique a grandement bénéficié des processeurs graphiques (GPU) pour accélérer l'entraînement et l'inférence. Cependant, ce succès n'a pas été reproduit pour l'optimisation combinatoire générale et exacte. Dans cet article, nous proposons de paralléliser sur GPU un solveur de programmation par contraintes, qui est une méthode générale et exacte basée sur la propagation de contraintes et le retour sur trace. Nous proposons un algorithme efficace de propagation de bornes d'intervalles entiers sur GPU pour résoudre des problèmes de contraintes discrètes. Notre approche décompose un réseau de contraintes en un réseau de contraintes ternaire optimisé pour les architectures GPU. Notre solveur de contraintes est significativement plus simple qu'un solveur de contraintes CPU optimisé, tout en étant compétitif par rapport aux solveurs séquentiels sur les problèmes du MiniZinc Challenge 2024.

Mots-clés

programmation par contraintes, GPU, solveur parallèle, cohérence de bornes

Abstract

Machine learning has tremendously benefited from graphics processing units (GPUs) to accelerate training and inference by several orders of magnitude. However, this success has not been replicated in general and exact combinatorial optimization. We explore GPU-accelerated constraint programming, a general and exact method based on constraint propagation and backtracking search. We propose an efficient integer interval bound propagation algorithm on GPUs for solving discrete constraint problems. Our approach decomposes a constraint network into a ternary constraint network optimized for GPU architectures. Our constraint solver is significantly simpler than an optimized CPU constraint solver, yet is competitive with optimized sequential solvers in the MiniZinc 2024 challenge.

Keywords

constraint programming, GPU, parallel solver, bound consistency

1 Introduction

L'apprentissage automatique a grandement bénéficié des processeurs graphiques (GPU) pour accélérer l'entraînement et l'inférence de plusieurs ordres de grandeur, par exemple dans la classification d'images [23]. Cependant, ce succès n'a pas été reproduit pour l'optimisation combinatoire générale et exacte. L'optimisation combinatoire parallélisée sur GPU a été appliquée aux algorithmes de méta-heuristiques [4, 16, 42, 22], ainsi qu'à l'optimisation exacte limitée à des problèmes spécifiques [18, 19, 1]. Une percée récente dans la programmation linéaire (PL) sur GPU a été rendue possible en utilisant un algorithme de résolution de PL plus exotique appelé *méthode hybride gradient primal-dual*, qui repose essentiellement sur des opérations de multiplication et d'addition matricielles [12, 2]. Cette méthode a été utilisée avec succès pour résoudre des programmes linéaires mixtes entiers dans la bibliothèque NVIDIA cuOpt. La programmation par contraintes est une méthode générale et exacte basée sur la propagation de contraintes et le retour sur trace [24]. Il n'existe que peu de tentatives pour paralléliser sur GPU la programmation par contraintes, soit en limitant l'expressivité du langage de contraintes aux variables à petits domaines [15], soit en accélérant uniquement les contraintes coûteuses tandis que tout le reste est exécuté sur le CPU [10, 43, 44]. Récemment, un nouveau modèle parallèle de calcul appliqué à la programmation par contraintes sur GPU a été proposé dans [41]. Ils prouvent formellement que la propagation de contraintes peut être parallélisée en place sans verrous et avec peu d'exigences matérielles, tant que les domaines satisfont une structure de treillis et que les propagateurs sont des fonctions monotones. Bien que la formalisation proposée soit générale et pour l'optimisation exacte, le prototype ne prend en charge que quelques contraintes appliquées à un problème d'ordonnancement. Un travail similaire spécialisé dans la propagation des domaines pour la PL a été proposé dans [38], mais repose sur des opérations atomiques de minimum et maximum.

L'implémentation proposée dans [41] est simple mais non optimisée pour les architectures GPU. Leur représentation des contraintes entraîne des accès mémoire non coalescés, un déséquilibre de charge, une divergence de threads et une pile non bornée ; autant d'éléments nuisibles à l'efficacité sur GPU [21]. Ces problèmes sont exacerbés lorsque l'on

considère un solveur de contraintes discret et général supportant de nombreuses contraintes (Section 3).

Pour relever ces défis, nous proposons un algorithme efficace de propagation de bornes d’intervalles entiers sur GPU. Le solveur proposé est général et peut être utilisé avec les formats de spécification de contraintes standard MiniZinc [27] et XCSP3 [9]. Notre approche décompose un réseau de contraintes en un *réseau de contraintes ternaire* avec un nombre réduit d’opérateurs (Section 4). Cette représentation ternaire nous permet d’optimiser les propagateurs sur GPU (Section 3). Comme la taille de la décomposition est jusqu’à 1000 fois plus grande que le réseau de contraintes initial, nous appliquons plusieurs techniques de prétraitement connues pour réduire sa taille (Section 5). L’augmentation médiane est de 4,8x en nombre de variables et de 4,3x en nombre de contraintes, bien que quatre problèmes (sur 90) soient restés plus de 100x plus grands.

Notre solveur de contraintes est nettement plus simple qu’un solveur de contraintes optimisé sur CPU. En particulier, on n’implémente pas de contraintes globales, d’apprentissage de nogoods, de génération paresseuse de clauses, de stratégies de redémarrage, de propagation événementielle, de restauration d’état basé sur le *trailing* ou recalcul, ni d’algorithme de cohérence de domaine. La plupart de ces optimisations sont considérées comme essentielles dans les solveurs de contraintes modernes [24]. L’architecture GPU nous a conduits à concevoir un solveur nettement plus simple :

- Cohérence de bornes en parallèle en utilisant une boucle de propagation similaire à AC1 — le premier algorithme de consistance utilisé en programmation par contraintes [25].
- Recalcul complet du nœud racine pour restaurer l’état [35].
- Recherche massivement parallèle [26] pour distribuer les nœuds de l’arbre de recherche sur les différents multiprocesseurs de flux du GPU (Section 2).

Malgré son algorithme peu sophistiqué, notre solveur est compétitif par rapport aux solveurs séquentiels sur les problèmes du *MiniZinc Challenge 2024*. En comparaison avec Choco [33], nous trouvons la même valeur objective sur 49% des instances, une meilleure valeur sur 23% et une pire sur 28%. En comparaison avec le meilleur solveur OR-Tools [32], nous sommes égaux dans 32% des cas, meilleurs dans 11% et moins bons dans 57%.

Dans son célèbre billet de blog *The Bitter Lesson*, Rich Sutton affirme que “les méthodes générales exploitant la puissance de calcul sont finalement les plus efficaces, et de loin” [39]. Cet article peut être considéré comme une première étape explorant cette thèse dans le contexte de la programmation par contraintes.

2 Contexte

2.1 Modèle de programmation CUDA

Nous présentons les composants de l’architecture GPU NVIDIA Hopper et le modèle de programmation CUDA afin de comprendre les sections suivantes. Pour illustrer

cette section, nous prenons l’exemple du système *NVIDIA GH200 Grace Hopper Superchip* [29] utilisé dans les expériences. Le GH200 combine un processeur Grace Arm avec 72 cœurs *Arm Neoverse V2* et un GPU H100 connecté par une bande passante élevée. Le H100 possède un total de 16896 cœurs regroupés en 132 multiprocesseurs de flux (SM) composés chacun de 128 cœurs. Il dispose d’une *mémoire globale* — la mémoire principale du GPU — de 96GB et d’un cache L2 de 50MB partagé entre tous les SM. Chaque SM possède son propre cache L1 de 256KB.

Le modèle de programmation CUDA suit la structure hiérarchique du matériel. Un *bloc* est un groupe de threads s’exécutant sur un seul SM. Dans la suite, nous fixons la taille d’un bloc à 256 threads, mais elle peut parfois varier en fonction de l’application (jusqu’à 1024 threads). Un *warp* divise un bloc en groupes de 32 threads, chaque bloc contenant donc 8 warps. Pour maximiser le parallélisme d’un warp, les threads doivent exécuter — autant que possible — les mêmes instructions sur des données multiples (SIMD). Les différents warps peuvent exécuter des instructions différentes en parallèle. Le pilote NVIDIA planifie les warps sur les SM en fonction, entre autres, de la disponibilité des données : lorsqu’un warp attend des données depuis la mémoire globale, un autre warp peut être planifié pour masquer la latence des données. Une fonction s’exécutant sur GPU est appelée un *kernel*. Pour une présentation plus exhaustive de CUDA et de la programmation GPU, nous renvoyons le lecteur à [30, 48].

Nous présentons maintenant trois défis de la programmation CUDA qui ne sont pas rencontrés dans la programmation CPU.

Regroupement des accès mémoire. Un thread lisant une valeur depuis la mémoire globale déclenche une transaction mémoire de 32, 64 ou 128 octets. Si tous les threads d’un warp lisent des entiers de 4 octets contigus, les 32 entiers peuvent être récupérés en une seule transaction mémoire de 128 octets, ce qui est appelé *regroupement des accès mémoire* (memory access coalescing). Dans le pire des cas, les 32 entiers sont répartis sur plus de 128 octets, ce qui conduit à 32 transactions mémoire. La différence d’efficacité entre les accès coalescés et non coalescés est généralement d’un ordre de grandeur, ce qui en fait une optimisation cruciale dans la programmation CUDA.

Divergence des threads. Les threads d’un warp exécutent une instruction commune à la fois. Considérons le programme suivant (où `threadIdx.x` est l’indice du thread par rapport à son bloc) :

```
int warpIdx = threadIdx.x % 32;
if(warpIdx < 16) P
else Q
```

La moitié des threads de chaque warp exécute *P* et l’autre moitié exécute *Q*. Les deux moitiés sont exécutées séquentiellement : au plus 16 threads sont actifs en même temps, car *P* et *Q* ne sont pas sur le même chemin d’exécution et n’ont donc aucune instruction commune. Dans la mesure du possible, ce code doit être réécrit pour éviter l’instruction conditionnelle en fusionnant *P* et *Q*.

Manque d'abstraction. En général, un kernel représente un algorithme parallélisé sur GPU, tel que un algorithme de tri ou la multiplication matrice-vecteur creuse (SpMV). Le CPU orchestre le flux principal de l'application en exécutant les kernels et les transferts de mémoire lorsque nécessaire. Le support de bibliothèque pour développer des kernels plus complexes est limité et, en particulier, la majeure partie de la bibliothèque standard C++ est inaccessible au sein d'un kernel. Une autre limitation est l'absence de primitives d'entrée/sortie (seul `printf` est supporté). Ainsi, le code CUDA typique reste relativement bas niveau avec une programmation de style C, notamment avec des pointeurs. Il est donc difficile d'écrire un logiciel de grande envergure tel qu'un solveur de contraintes dans un seul kernel. Néanmoins, la plupart de la syntaxe et de la sémantique de C++20 est supportée et une nouvelle bibliothèque standard compatible CUDA (`libc++`) est en cours de développement par NVIDIA [31], bien qu'elle manque encore de nombreuses structures de données utiles comme `vector`. En attendant, nous avons développé une bibliothèque de structure de données compatible CUDA, appelé *cuda-battery* et disponible en ligne¹.

2.2 Programmation par Contraintes

Dans la suite, nous considérons la programmation par contraintes sur des variables entières uniquement. Soit X un ensemble fini de variables et C un ensemble fini de contraintes. Pour chaque contrainte $c \in C$, nous notons par $scp(c) \subseteq X$ sa *portée*, c'est-à-dire l'ensemble des variables libres de c . Un *réseau de contraintes* est une paire $P = \langle \rho, C \rangle$ telle que $\rho \in X \rightarrow \mathcal{P}(\mathbb{Z})$ est le *domaine* des variables et $\forall c \in C, scp(c) \subseteq X$. Une *affectation* est une fonction $asn : X \rightarrow \mathbb{Z}$, et nous notons l'ensemble de toutes les affectations par **Asn**. L'ensemble des solutions d'une contrainte est donné par $rel(c) \subseteq \mathbf{Asn}$. L'ensemble des solutions d'un réseau de contraintes est :

$$sol^b(\rho, C) \triangleq \{asn \in \mathbf{Asn} \mid \forall c \in C, asn \in rel(c) \wedge \forall x \in X, asn(x) \in \rho(x)\}$$

Sans perte de généralité, nous représentons le domaine des variables à l'aide d'intervalles. Soit $I = \{[\ell, u] \mid \ell \in \mathbb{Z} \cup \{-\infty\}, u \in \mathbb{Z} \cup \{\infty\}, \ell \leq u\} \cup \{\perp\}$ l'ensemble des intervalles ordonnés par inclusion avec un élément spécial $\perp$ représentant l'intervalle vide. Dans la suite, nous manipulons exclusivement des intervalles, et donc la fonction de domaine est définie par $d \in X \rightarrow I$. Nous notons $\mathbf{D}$ l'ensemble de toutes les fonctions de domaine $X \rightarrow I$, ordonné point par point ($d \leq d' \Leftrightarrow \forall x \in X, d(x) \subseteq d'(x)$). Soit $[\ell, u]$ un intervalle, l'ensemble des valeurs dans l'intervalle est donné par $\gamma([\ell, u]) \triangleq \{v \in \mathbb{Z} \mid \ell \leq v \leq u\}$.

Un propagateur d'intervalle est une fonction $p_c : \mathbf{D} \rightarrow \mathbf{D}$ où $c \in C$ est une contrainte. Pour $d \in \mathbf{D}$, un propagateur est réducteur ($p_c(d) \leq d$), monotone ($d \leq d' \Rightarrow p_c(d) \leq p_c(d')$) et correct ($sol(d, \{c\}) \subseteq sol(p_c(d), \{c\})$). Il est également complet pour les intervalles singletons : lorsque $\forall x \in scp(c), \exists v \in \mathbb{Z}, d(x) = [v, v]$, alors $sol(d, \{c\}) \supseteq sol(p_c(d), \{c\})$.

La propagation de contraintes consiste à trouver le plus grand point fixe d'un ensemble de propagateurs $\{p_1, \dots, p_n\}$ sur un domaine $d \in X \rightarrow I$. Tant que les propagateurs sont exécutés de manière équitable, leur ordre d'exécution n'a pas d'importance et le même plus grand point fixe est toujours atteint [3]. Ce fait a été utilisé pour concevoir divers *algorithmes de propagation* afin d'accélérer le calcul du point fixe [36, 40].

Comme la propagation de contraintes est correcte mais incomplète en général, elle doit être entrelacée avec une procédure de recherche. Soit $split \in \mathbf{D} \rightarrow \mathcal{P}(\mathbf{D})$ une fonction de branchement strictement réductrice ($\forall d \in \mathbf{D}, \forall d' \in split(d), d' < d$), correcte et complète ($\forall d \in \mathbf{D}, \bigcup \{sol(d', C) \mid d' \in split(d)\} = sol(d, C)$), nous définissons l'algorithme de résolution par *propagation et recherche* comme suit.

```

function SOLVE( $d, \{c_1, \dots, c_n\}$ )
   $d \leftarrow \mathbf{gfp}_d p_{c_1} \circ \dots \circ p_{c_n}$ 
  if  $\forall x \in X, d(x) = [v, v]$  then return  $\{d\}$ 
  else if  $\exists x \in X, d(x) = \perp$  then return  $\{\}$ 
  else
     $\langle d_1, \dots, d_n \rangle \leftarrow split(d)$ 
    return  $\bigcup_{i=1}^n solve(d_i, C)$ 
  end if
end function

```

Le plus grand point fixe paramétrique $\mathbf{gfp}_x f$ d'une fonction f est équivalent à $\mathbf{gfp} (f \circ \lambda y.x \cap y)$, c'est-à-dire que nous commençons le calcul du point fixe à partir de l'élément x . Il est bien connu que l'algorithme `solve` est une procédure de résolution correcte et complète, voir par exemple [24, 40]. Le résultat reste valable même en présence d'intervalles infinis tant qu'ils deviennent finis après un nombre fini d'étapes de propagation.

Programmation par contraintes parallèle. Le seul type de parallélisme ayant fait ses preuves dans les solveurs de programmation par contraintes consiste à diviser la recherche entre les threads (par exemple, en assignant un thread par sous-problème $d_1, \dots, d_n$), ou à la répéter différemment dans une approche portfolio (par exemple, en utilisant différentes fonctions *split*) [17]. Dans les deux cas, les threads travaillent sur une copie locale du problème. Cela est efficace et facile à implémenter car les threads nécessitent peu de synchronisation et de communication, comme illustré par l'algorithme de recherche massivement parallèle (EPS) [26].

Représentation des propagateurs. Il est possible de construire un nombre infini de contraintes, prenons par exemple la séquence $\langle x_1 = x_2, x_1 = x_2 \odot_1 x_3, x_1 = x_2 \odot_1 x_3 \odot_2 x_4, \dots \rangle$ où chaque $\odot_i$ est un opérateur arithmétique quelconque. Par conséquent, il est impossible de créer statiquement un propagateur par contrainte possible. Historiquement, les solveurs ont limité leurs langages de contraintes afin d'éviter la mise en œuvre d'un trop grand nombre de propagateurs [45, 13, 8]. Les contraintes plus complexes doivent être réécrites en contraintes prises en charge, soit automatiquement, soit par l'utilisateur. L'approche des *indexicaux* est un moyen particulièrement

1. <https://github.com/lattice-land/cuda-battery>

concis de spécifier des propagateurs à l'aide de quelques constructions primitives [11, 46].

Cependant, la décomposition des contraintes en contraintes primitives augmente le nombre de variables auxiliaires et de propagateurs. Il a été démontré que cela nuit aux performances du solveur [37, 14]. L'idée clé est donc d'implémenter un propagateur en tant qu'interpréteur sur l'arbre syntaxique abstrait (AST) de la contrainte. Le propagateur parcourt récursivement l'AST pour évaluer chaque nœud et calculer le domaine de chaque sous-expression de la contrainte. On parle de *propagateur basé sur les vues* dans les travaux récents [37, 14], mais une technique similaire avait déjà été utilisée dans l'algorithme de cohérence HC-4 [7].

Les solveurs de contraintes modernes tels que Choco et OR-Tools implémentent la propagation basée sur les vues en utilisant l'héritage pour représenter l'AST, et le polymorphisme de sous-type pour évaluer l'arbre. Pour éviter la surcharge causée par le sous-typage dynamique, le polymorphisme paramétrique a également été utilisé [14, 37], mais cela présente l'inconvénient que le solveur doit être recompilé pour chaque nouveau problème de contrainte traité.

3 Propagation Efficace de Bornes sur GPU

La représentation basée sur les vues entraîne des accès mémoire non coalescés, un déséquilibre de charge, une divergence de threads et une pile non bornée ; autant d'éléments nuisibles à l'efficacité sur GPU [21].

Pour comprendre ces problèmes, nous considérons que les propagateurs sont stockés dans un tableau de pointeurs de la forme `Propagator** p` où `Propagator` est une interface. Comme nous considérons des propagateurs peu coûteux en calcul pour des contraintes arithmétiques simples (et non des contraintes globales), une stratégie parallèle consiste à exécuter un propagateur par thread `gfpd pc1 || ... || pcn`. Il est garanti que le même plus grand point fixe que celui calculé en Section 2 est finalement atteint [41].

Les lectures mémoires initiales sont coalescées lorsque deux threads accèdent à `p[0]` et `p[1]`. Cependant, comme `*p[0]` et `*p[1]` ne sont pas nécessairement contigus en mémoire, les lectures futures peuvent être non coalescées. Il est possible d'améliorer la situation en aplatissant la représentation des propagateurs dans un tableau de nœuds, de manière similaire à ce qui est utilisé pour représenter les AST dans les compilateurs [47]. Le nœud racine des deux propagateurs sont stockés de manière contigües, puis les deux nœuds de gauche, et ainsi de suite. Lorsque deux propagateurs ont des AST de formes différentes, nous devons stocker la longueur des enfants, ou pour maintenir la coalescence mémoire, utiliser un remplissage similaire au format ELL pour les matrices creuses [6]. Mais cela conduit toujours à un déséquilibre de charge car certains threads peuvent devenir inactifs, et le temps d'exécution d'un groupe de propagateurs parallèles est borné par le plus lent d'entre eux. De plus, si les opérandes des nœuds ne sont

pas du même type (par exemple, une addition et une multiplication), les threads divergent et l'évaluation des nœuds devient séquentielle. Enfin, lors de l'évaluation de l'AST, des variables locales sont créées et empilées pour stocker les intervalles des sous-expressions. Les fonctions récursives empêchent le compilateur CUDA de faire l'*inlining* et de calculer automatiquement la taille de la pile.

Notre solution pour résoudre ces problèmes consiste à réécrire le réseau de contraintes en un *réseau de contraintes ternaire*, c'est-à-dire un réseau de contraintes ne contenant que des contraintes d'arité 3 telles que $x = y + z$ et $b = (x \leq y)$. Nous introduisons la *forme normale ternaire* (TNF), qui est une représentation optimisée des contraintes ternaires pour GPU.

Définition 3.1. Une contrainte est en TNF si elle est de la forme $x = y \odot z$ où $x, y, z \in X$ sont des variables et $\odot \in OP = \{+, *, /, \text{mod}, \min, \max, =, \leq\}$.

Les contraintes unaires de la forme $x = k$, $x \leq k$ et $x \geq k$ où $x \in X$ et $k \in \mathbb{Z}$ ne sont pas représentées comme des contraintes mais directement comme des domaines des variables.

La définition de l'ensemble OP pourrait être différente. Par exemple, la contrainte $x = \min(y, z)$ peut être réécrite sous la forme $-x = \max(-y, -z)$, ce qui, une fois décomposé en contraintes ternaires, donne $\exists x', \exists y', \exists z', x' = \max(y', z') \wedge x' = \text{zero} - x \wedge y' = \text{zero} - y \wedge z' = \text{zero} - z$, avec $d(\text{zero}) = [0, 0]$.

Cela permet de réduire la taille de OP , mais introduit trois nouvelles variables et contraintes. Comme nous le verrons, les opérateurs $\min$ et $\max$ sont particulièrement importants car ils permettent aussi d'encoder la disjonction et la conjonction, et il est donc préférable de les conserver.

L'absence de soustraction est justifiée par la sémantique relationnelle des contraintes, car on peut réécrire $x = y - z$ sous la forme $y = x + z$ sans perte de précision.

Ce petit ensemble d'opérateurs est suffisamment expressif pour résoudre des instances générales de MiniZinc sur variables entières. Une question ouverte est de savoir s'il existe un meilleur ensemble d'opérateurs OP pour l'exécution sur GPU.

Nous montrons en Section 4 comment réécrire un réseau de contraintes en un réseau de contraintes ternaire utilisant uniquement des contraintes en TNF. Nous discutons maintenant comment la TNF répond aux quatre problèmes mentionnés précédemment.

Coalescence des accès mémoire. Un propagateur implémentant une contrainte en TNF peut être représenté par une structure de 16 octets contenant les indices de trois variables et le type d'opérateur.

```
struct Bytecode {
    OP op; // OP est une énumération entière.
    int x;
    int y;
    int z;
};
```

Les propagateurs sont stockés dans un tableau `Bytecode*` `bc`, et chaque thread d'un warp charge en mémoire des propagateurs contigus. Un warp contient 32 threads et une transaction mémoire couvre 128 octets. Par conséquent, seules 4 transactions mémoire sont nécessaires ($32 * 16 / 128$) pour lire tous les propagateurs d'un warp.

De plus, nous obtenons un bon équilibrage de charge, car aucun chemin d'exécution n'est significativement plus long qu'un autre — toutes les opérations sont définies par de l'arithmétique sur intervalles, et donc rapides.

On pourrait argumenter que davantage de transactions mémoire sont nécessaires car il faut lire 4 entiers par propagateur, nécessitant ainsi 4 transactions pour `op`, puis 4 pour `x`, et ainsi de suite, menant à 16 transactions. Nous évitons ce comportement en effectuant un chargement vectorisé de 4 entiers en une seule instruction PTX (le langage assembleur de CUDA). Pour cela, il suffit de convertir `Bytecode` en type spécial CUDA `int4` comme suit :

```
__device__ Bytecode load(int i) {
    int4 b4 = reinterpret_cast<int4*>(bc)[i];
    return *reinterpret_cast<bytecode_type*>(&b4);
}
```

À l'aide du profileur CUDA, nous avons pu vérifier que cette fonction est bien compilée en une seule instruction PTX `ld.global.v4.s32`, sans variable temporaire supplémentaire.

Notons qu'une représentation plus grande des propagateurs (par exemple, 20 octets) nous empêcherait de vectoriser les lectures, car 16 octets est la taille maximale supportée.

Minimisation de la divergence des threads. Une fois le bytecode `b` en mémoire, nous devons lire le domaine des trois variables `b.x`, `b.y` et `b.z` dans des variables locales et propager en fonction de l'opérateur `b.op`. Nous remarquons que les lectures des domaines des variables peuvent être non-coalescés, car les propagateurs n'accèdent pas nécessairement à des variables contigües en mémoire. Nous laissons ouverte la question de savoir comment coalescer les lectures des domaines des variables, et si cela est possible ou non.

Une première optimisation consiste à trier lexicographiquement le tableau de bytecodes selon $\langle op, y, x, z \rangle$. Cela réduit la divergence car les propagateurs ayant les mêmes opérateurs sont exécutés ensemble. De plus, nous trions les variables afin de réduire le nombre de transactions mémoire — l'ordre y, x, z a donné les meilleurs résultats.

Une deuxième optimisation est de choisir un petit ensemble d'opérateurs OP afin de minimiser la divergence des threads, mais aussi de rendre le code plus court, ce qui favorise davantage l'inlining et nécessite moins de chargements d'instructions. Bien que OP contienne 8 opérateurs distincts, la fonction de propagation peut diverger en 13 points en raison des opérateurs $\leq$ et $=$ pouvant être niés (par exemple, $0 = (y = z)$ modélise $y \neq z$) et réifiés. Nous effectuons une analyse des opérateurs utilisés dans différents problèmes de contraintes en Section 6.

Une troisième optimisation consiste à représenter les

constantes comme des variables. Cela permet de représenter les contraintes ternaires avec seulement 16 octets comme montré précédemment, mais aussi d'éviter la divergence dans les feuilles de l'AST où un thread pourrait charger une constante et l'autre une variable.

Sans récursion. Étant donné que les contraintes ont une représentation régulière, les propagateurs peuvent être implémentés comme des fonctions sans récursion en utilisant quelques instructions `switch` sur `b.op`. Cela permet au compilateur CUDA de réaliser un inlining agressif, notamment pour les fonctions d'arithmétique sur intervalles.

Point fixe par warp. Les propagateurs exécutés dans un warp traitent souvent des contraintes partageant des variables. Ce regroupement est presque accidentel et dû à la manière dont nous écrivons les modèles de contraintes, mais aussi à la décomposition de chaque contrainte qui est spatialement proche dans le tableau de bytecodes. Avec cette observation en tête, nous proposons une nouvelle formulation de point fixe par warp :

$$\mathbf{gfp}_d (\lambda x. \mathbf{gfp}_x p_{c_1} \parallel \dots \parallel p_{c_{32}}) \parallel \dots \parallel (\lambda x. \mathbf{gfp}_x p_{c_{n-32}} \parallel \dots \parallel p_{c_n})$$

Chaque warp atteint un point fixe local avant de passer aux 32 propagateurs suivants. D'un point de vue algorithmique, cela réduit le nombre total d'appels aux propagateurs. D'un point de vue architecture matériel, cela permet de réutiliser le bytecode déjà chargé dans le cache et les registres. Il est important de noter que contrairement à ce que suggère la formalisation mathématique ci-dessus, dans notre implémentation, les domaines des variables sont partagés entre les warps. En effet, nous lisons les domaines à chaque itération pour permettre aux warps d'échanger des informations pendant le calcul de leur point fixe local.

Bien que cette définition mathématique diffère de l'implémentation, elle est plus simple à utiliser dans notre contexte et a été démontrée équivalente [41]. La boucle de point fixe par warp est montrée plus efficace en Section 6. Elle est également correcte, comme démontré par la proposition suivante.

Proposition 3.1. *Les deux plus grands points fixes suivants sont égaux :*

$$\mathbf{gfp}_d p_{c_1} \parallel \dots \parallel p_{c_n} = \mathbf{gfp}_d (\lambda x. \mathbf{gfp}_x p_{c_1} \parallel \dots \parallel p_{c_{32}}) \parallel \dots \parallel (\lambda x. \mathbf{gfp}_x p_{c_{n-32}} \parallel \dots \parallel p_{c_n})$$

Démonstration. L'ordre dans lequel les propagateurs mettent à jour la mémoire dépend de la stratégie de planification des threads sous-jacente. Cependant, il a été démontré que le plus grand point fixe parallèle est toujours le même, indépendamment de la stratégie de planification [41]. Le point fixe par warp impose essentiellement une contrainte sur la stratégie de planification en exécutant des groupes de 32 propagateurs jusqu'à ce qu'un point fixe local soit atteint. Par conséquent, il existe une stratégie de planification des propagateurs pour $\mathbf{gfp}_d p_{c_1} \parallel \dots \parallel p_{c_n}$

qui correspond à celle utilisée pour le point fixe par warp. Puisque le plus grand point fixe est indépendant de la stratégie de planification, les deux points fixes doivent être égaux. $\square$

4 Réseau de Contrainte Ternaire

Soit $\langle d \in X \rightarrow I, C \rangle$ un réseau de contraintes. Nous introduisons la fonction $tcn(d, C) = \langle d', C' \rangle$ qui réécrit un réseau de contraintes en un réseau de contraintes ternaire où chaque contrainte est en TNF. Le langage de contraintes considéré pour C est suffisant pour prendre en charge 95 instances entières du MiniZinc Challenge 2024—les 5 restantes utilisant des variables ensemblistes non supportées pour le moment. La fonction tcn ne fait qu'ajouter des nouvelles variables et par conséquent nous avons $dom(d) \subseteq dom(d')$ où dom est le domaine de la fonction d .

Pour plus de clarté, nous notons $\langle b \ ? \ X \ ? \ Y \rangle$ la fonction qui retourne X si l'expression b est vraie et Y sinon. Nous utilisons également l'expression $let \ x = E \ in \ E'$ qui associe le résultat de l'évaluation de l'expression E à la variable x , et retourne l'évaluation de l'expression E' où E' peut utiliser x . Cela s'étend naturellement au cas où E retourne un tuple de valeurs, par exemple $let \ x, y = E \ in \ E'$.

Nous définissons d'abord plusieurs fonctions pour étendre un réseau de contraintes avec une nouvelle variable $x \notin dom(d)$, et pour mettre à jour une variable déjà présente dans d :

$$\begin{aligned} extend(d, x, [\ell, u]) &= (d \cup \{x \mapsto [\ell, u]\}, x) \\ extend(d, x) &= extend(d, x, [-\infty, \infty]) \\ extend(d) &= extend(d, fresh(d)) \\ extend_{\mathbb{B}}(d) &= extend(d, fresh(d), [0, 1]) \\ extend_{co}(d, k) &= \\ &\quad let \ c = _CONSTANT_ \langle k < 0 \ ? \ mk : k \rangle \ in \\ &\quad \langle c \in dom(d) \ ? \ (d, c) : extend(d, c, [k, k]) \rangle \\ update(d, x, [\ell, u]) &= \\ &\quad \{y \mapsto \langle x = y \ ? \ [\ell, u] \cap d(y) : d(y) \rangle \mid y \in dom(d)\} \end{aligned}$$

La fonction $fresh(d)$ retourne un nom de variable x tel que $x \notin dom(d)$. Nous utilisons la fonction $extend_{\mathbb{B}}$ pour créer une nouvelle variable booléenne. Afin d'éviter de créer plusieurs variables représentant une même constante, nous générons un nom unique pour chaque constante. Par exemple, $_CONSTANT_5$ est le nom de la variable correspondant à la constante 5, et $_CONSTANT_m1$ pour la constante -1 . Nous supposons qu'aucun nom de variable dans le réseau de contraintes initial n'est de cette forme. La fonction $update(d, x, [\ell, u])$ intersecte le domaine $[\ell_x, u_x]$ d'une variable x avec un intervalle $[\ell, u]$, où l'intersection d'intervalles est définie par $[\ell_x, u_x] \cap [\ell, u] \triangleq [\max\{\ell_x, \ell\}, \min\{u_x, u\}]$.

La fonction $tcn(d, C) = \langle d', C' \rangle$ réécrit chaque contrainte

en TNF :

$$\begin{aligned} tcn(d, \{\}) &= \langle d, \{\} \rangle \\ tcn(d, \{c_1, c_2, \dots, c_n\}) &= \\ &\quad let \ d, T = tcn(d, \{c_2, \dots, c_n\}) \ in \\ &\quad let \ d, U, x = tnf(d, c_1) \ in \\ &\quad \langle update(d, x, [1, 1]), T \cup U \rangle \end{aligned}$$

Chaque contrainte $c \in C$ est réécrite en un ensemble T de contraintes en TNF, éventuellement avec de nouvelles variables dans d , à l'aide de la fonction tnf . La variable x est une variable booléenne qui réifie la contrainte c . Comme les contraintes doivent toutes être satisfaites, nous devons fixer cette variable x à 1 pour activer la contrainte, ce qui est fait en mettant à jour son domaine.

Nous introduisons la fonction récursive $tnf(d, t) = (d', T, x)$ qui réécrit un terme ou une contrainte t en un ensemble T de contraintes en TNF. Le résultat de l'expression t (ou son statut de réification s'il s'agit d'une contrainte) est stocké dans la variable x . Nous considérons d'abord les cas de base (variables et constantes) et les contraintes unaires (négation arithmétique $-$, appartenance à un ensemble $\in$ et valeur absolue abs).

Les variables sont simplement retournées, car toute variable apparaissant dans une contrainte doit déjà être présente dans d . Une nouvelle variable est créée pour chaque constante distincte k en utilisant $extend_{co}$ défini précédemment. Les contraintes unaires sont réécrites en utilisant des contraintes ternaires équivalentes.

Pour l'appartenance à un ensemble, nous avons besoin de la fonction $itvs(S)$ qui transforme un ensemble S en un ensemble d'intervalles. Par exemple, $itvs(\{1, 2, 3, 5\}) = \{[1, 3], [5, 5]\}$. La stratégie de réécriture est toujours la même : nous réécrivons d'abord les paramètres de la fonction ou du prédicat, puis nous assemblons les résultats pour réécrire l'expression courante.

$$\begin{aligned} tnf(d, x) &= (d, \{\}, x) \\ tnf(d, -t) &= tnf(d, 0 - t) \\ tnf(d, t \in S) &= \\ &\quad let \ d, T, x = tnf(d, t) \ in \\ &\quad let \ d = update(d, x, [\min S, \max S]) \ in \\ &\quad let \ d, U, y = \\ &\quad \quad tnf(d, \bigvee_{[\ell, u] \in itvs(S)} (x \geq \ell \wedge x \leq u)) \ in \\ &\quad (d, T \cup U, y) \\ tnf(d, k) &= \\ &\quad let \ d, x = extend_{co}(d, k) \ in \\ &\quad (d, \{\}, x) \\ tnf(d, abs(t)) &= \\ &\quad let \ d, T, x = tnf(d, t) \ in \\ &\quad let \ d, U, y = tnf(d, 0 - x) \ in \\ &\quad let \ d, V, z = tnf(d, \max(x, y)) \ in \\ &\quad let \ d = update(d, z, [0, \infty]) \\ &\quad tnf(d, T \cup U \cup V, z) \end{aligned}$$

La réécriture des opérateurs binaires $\odot \in OP$ est factorisée dans une fonction unique. Lorsque l'opérateur est booléen

($\leq, =$), le résultat est stocké dans une variable booléenne, sinon dans une variable entière.

Les autres opérateurs peuvent être réécrits en expressions utilisant les opérateurs *OP*. En particulier, comme la soustraction est l'inverse de l'addition, nous avons $x = y - z \Leftrightarrow y = x + z$. Ce n'est pas le cas pour la multiplication et la division sur les entiers, c'est pourquoi nous conservons les deux opérateurs. Nous réécrivons l'opérateur $\neq$ en utilisant la négation de l'égalité. Comme nous pouvons être dans un contexte réifié, nous ne pouvons pas simplement ajouter la contrainte $TNF\ zero = (x = y)$, c'est pourquoi nous délégons la réécriture à l'équivalence logique.

```
tnf(d, t1 ⊙ t2) =
  let d, x =
    (⊙ ∈ {≤, =} ? extendℤ(d) : extend(d)) in
  let d, T, y = tnf(d, t1) in
  let d, U, z = tnf(d, t2) in
  (d, T ∪ U ∪ {x = y ⊙ z}, x)

tnf(d, t1 ≠ t2) =
  let d, zero = extendco(d, 0) in
  tnf(d, zero ⇔ t1 = t2)

tnf(d, t1 − t2) =
  let d, x = extend(d) in
  let d, T, y = tnf(d, t1) in
  let d, U, z = tnf(d, t2) in
  (d, T ∪ U ∪ {y = x + z}, x)

tnf(d, t1 ≥ t2) = tnf(d, t2 ≤ t1)
tnf(d, t1 > t2) = tnf(d, t2 ≤ t1 − 1)
tnf(d, t1 < t2) = tnf(d, t1 ≤ t2 − 1)
```

La dernière étape consiste à compiler les connecteurs logiques en contraintes TNF. Les formules logiques peuvent apparaître dans des expressions arithmétiques, par exemple $(x \neq y \wedge x \neq z) + (y \neq z) \geq 1$, et les variables entières dans les formules logiques, par exemple $((x + w) \vee y) = z$. Dans un contexte booléen, nous interprétons la valeur d'un intervalle $[\ell, u]$ comme étant *true* si et seulement si $0 \notin [\ell, u]$, et *false* si $\ell = u = 0$. Cependant, si la contrainte est réifiée, nous devons associer *true* à 1 et non à n'importe quelle valeur. Comme nous compilons $\vee$ en *min* et $\wedge$ en *max*, nous devons nous assurer que le résultat des connecteurs logiques se situe dans l'intervalle $[0, 1]$.

La fonction *booleanize* mappe le domaine d'une expression apparaissant dans un contexte booléen à une variable booléenne. Comme optimisation, nous ne connectons x à une variable booléenne que si ce n'est pas déjà une variable booléenne, c'est-à-dire si son domaine ne se situe pas dans l'intervalle $[0, 1]$. Cette fonction est ensuite utilisée pour réécrire tous les connecteurs logiques.

Il est à noter que nous ne pouvons pas directement réécrire $t_1 \text{ xor } t_2$ en $t_1 \neq t_2$, sinon des expressions telles que $[1, 1] \text{ xor } [2, 2]$ seraient évaluées à *true*, ce qui, selon notre sémantique, n'est pas correct car à la fois $[1, 1]$ et $[2, 2]$ doivent être associés à *true*.

```
booleanize(d, t) =
  let d, T, x = tnf(d, t) in
  if ¬(d(x) ≤ [0, 1]) then
    let d, U, b = tnf(d, x ≠ 0) in
    (d, T ∪ U, b)
  else (d, T, x)

tnf(d, ¬t) = tnf(d, t = 0)

tnf(d, t1 ⇒ t2) = tnf(d, ¬t1 ∨ t2)

tnf(d, t1 ∧ t2) =
  let d, b = extendℤ(d) in
  let d, T, b1 = booleanize(d, t1) in
  let d, U, b2 = booleanize(d, t2) in
  (d, T ∪ U ∪ {b = min(b1, b2)}, b)

tnf(d, t1 ∨ t2) =
  let d, b = extendℤ(d) in
  let d, T, b1 = booleanize(d, t1) in
  let d, U, b2 = booleanize(d, t2) in
  (d, T ∪ U ∪ {b = max(b1, b2)}, b)

tnf(d, t1 ⇔ t2) =
  let d, b = extendℤ(d) in
  let d, T, b1 = booleanize(d, t1) in
  let d, U, b2 = booleanize(d, t2) in
  (d, T ∪ U ∪ {b = (b1 = b2)}, b)

tnf(d, t1 xor t2) =
  let d, T, b1 = booleanize(d, t1) in
  let d, U, b2 = booleanize(d, t2) in
  let d, V, b = tnf(d, b1 ≠ b2) in
  (d, T ∪ U ∪ V, b)
```

Nous prouvons maintenant que chaque réseau de contraintes $\langle d, C \rangle$ est équivalent à sa réécriture $tcn(d, C)$, c'est-à-dire qu'il possède exactement le même ensemble de solutions. Comme nous introduisons de nouvelles variables, nous devons restreindre les solutions à celles définies sur les variables initiales. Soit $asn : X \rightarrow \mathbb{Z}$ une affectation et Y un sous-ensemble de X , sa restriction $asn|_Y$ est définie par $\{y \mapsto asn(y) \mid y \in Y\}$. Nous étendons la restriction à l'ensemble des affectations A comme suit : $A|_Y = \{asn|_Y \mid asn \in A\}$.

Proposition 4.1 (Correction et complétude). *Soit $\langle d, C \rangle$ un réseau de contraintes défini sur l'ensemble de variables X , alors $sol(d, C) = sol(tcn(d, C))|_X$.*

Proposition 4.2 (Unicité). *De plus, nous avons une bijection entre les deux ensembles de solutions : $|sol(d, C)| = |sol(tcn(d, C))|$.*

Ces affirmations peuvent être prouvées par induction structurée sur la formule, montrant qu'à chaque étape l'ensemble des solutions est préservé.

5 Prétraitement du Réseau de Contrainte Ternaire

Nous suivons des techniques de prétraitement standards que nous spécialisons pour les réseaux de contraintes ter-

naires [34, 28]. Le prétraitement a un impact substantiel sur les performances comme le montrent les expériences. L’objectif du prétraitement est essentiellement de supprimer des variables et des contraintes. Avant de présenter notre algorithme de prétraitement, nous définissons une structure pour détecter les variables équivalentes.

Nous cherchons à trouver une partition E de X telle que chaque composante Y de E représente un ensemble de variables équivalentes. Plus précisément, toutes les paires de variables $x, y \in Y$ sont connectées par une contrainte d’égalité $x = y$. Nous écrivons $[x]_E \in E$ pour désigner la classe d’équivalence de x dans E . Nous supposons que les variables sont totalement ordonnées (par exemple, par un indexage) et nous choisissons $\min Y$ comme variable représentative de la classe d’équivalence Y . Les classes d’équivalence sont découvertes par diverses techniques de prétraitement. Initialement, nous supposons que les variables sont toutes distinctes, ce qui est donné par la partition $\text{init}(X) \triangleq \{\{x\} \mid x \in X\}$. Nous ajoutons une égalité de variables $x = y$ en supprimant les deux classes d’équivalence $[x]_E$ et $[y]_E$ de E , puis en ajoutant leur union dans la partition. Le domaine d’intervalle d’une variable $x \in X$ dans une classe d’équivalence Y est l’intersection des domaines de toutes les variables de Y , défini comme $d_E(x) \triangleq \bigcap_{y \in [x]_E} d(y)$.

Nous appliquons sept fonctions de prétraitement :

- *Propagation* pour réduire le domaine des variables.
- *Simplification algébrique* pour éliminer les contraintes dont les solutions peuvent être directement exprimées dans le domaine des variables. Elle détecte également les équivalences entre les variables et affine la partition E .
- *Élimination des sous-expressions communes* pour détecter les contraintes ternaires ayant la même expression à droite de l’égalité. Par exemple, considérons $a = y + z$ et $b = y + z$, la procédure élimine l’une des deux contraintes et fusionne les classes d’équivalence de a et b .
- *Fusion des domaines* des variables dans les classes d’équivalence. Cela est particulièrement utile pour la propagation qui ne connaît pas les classes d’équivalence.
- *Élimination des contraintes redondantes* pour supprimer les contraintes qui sont déjà impliquées par d . Par exemple, si $d(x) = [1, 2]$ et $d(y) = [2, 3]$, alors la contrainte ternaire $1 = (x \leq y)$ est toujours vraie, indépendamment de l’évolution de d .
- *Renommage des variables* pour utiliser une variable unique par classe d’équivalence.
- *Élimination des variables inutiles* pour supprimer les variables qui ne sont dans le domaine d’aucune contrainte. Nous conservons les variables ayant un domaine vide afin de pouvoir détecter l’insatisfiabilité.

Nous donnons un exemple de l’algorithme de prétraitement sur le réseau de contraintes $\langle d, \{x = y + z, w = y + z, x = (y = z)\} \rangle$ avec $d(x) = [0, 1]$, $d(w) = [1, 2]$ et toutes les autres variables ayant pour domaine $[-\infty, \infty]$.

La propagation est incapable de supprimer des valeurs des

domaines et la simplification algébrique ne détecte aucune contrainte d’égalité puisque x pourrait être égal à 0. Ensuite, l’élimination de sous-expressions communes détecte l’égalité $x = w$, modifie la partition en $\{\{x, w\}, \{y\}, \{z\}\}$ et supprime la contrainte $x = (y = z)$.

La propagation reste inefficace, mais cette fois la simplification algébrique est capable de détecter l’égalité $y = z$ car $d_E(x) = d(x) \cap d(w) = [1, 1]$, et la partition est mise à jour en $\{\{x, w\}, \{y, z\}\}$. Ensuite, la simplification algébrique réécrit la contrainte $w = y + z$ en $w = y * 2$. Cela permet à l’étape de propagation de réduire le domaine de y à $\perp$, détectant ainsi effectivement que le problème est insatisfiable.

À ce stade, le réseau de contraintes est $\langle d, \{w = y * 2\} \rangle$ et le prétraitement termine. Dans cette situation, le problème a été complètement résolu, et détecté comme insatisfiable, par le prétraitement.

6 Évaluation Expérimentale

Nous évaluons notre approche sur 90/100 instances du MiniZinc Challenge 2024. Nous supprimons 5 instances du problème *harmony* contenant des variables d’ensemble (actuellement non prises en charge) et 5 instances supplémentaires du problème *yumi-dynamic* pour lesquelles notre décomposition est trop volumineuse pour être résolue en parallèle sur GPU (nécessite plus de 96GB). Les 90 instances restantes peuvent être résolues avec notre approche. Pour toutes les instances, le modèle MiniZinc est converti en un format plus simple appelé FlatZinc, comme cela est fait pendant le MiniZinc Challenge. Il y a une instance qui est détectée comme insatisfiable lors de la conversion en FlatZinc et que nous supprimons des expériences lorsqu’elle n’est pas pertinente.

Notre solveur GPU, Turbo v1.2.8², est exécuté sur le superprocesseur NVIDIA GH200 décrit dans la Section 2. Choco v4.10.18 et OR-Tools v9.9 sont exécutés sur un processeur AMD Epyc ROME 7H12 avec 64 cœurs cadencés à 2,6GHz. Nous fixons la limite de temps pour chaque instance à 20 minutes.

Pour assurer l’équité, nous créons les sous-problèmes à résoudre en parallèle en suivant la même stratégie de recherche que celle utilisée pour la résolution. Cela permet à notre algorithme parallèle d’explorer un arbre de recherche similaire aux solveurs séquentiels, ce qui nous désavantage car cela conduit à des blocs inactifs dans 20% des instances en raison d’un déséquilibre de charge. La propagation et la recherche sont entièrement exécutées sur le GPU — le CPU est limité à l’analyse, au prétraitement et à l’affichage du résultat final. La création des sous-problèmes est incluse dans la limite de temps, ainsi que les décompositions FlatZinc et TCN.

Analyse des réseaux de contrainte ternaire. Comme nous ne prenons en charge aucune contrainte globale, la conversion de MiniZinc en FlatZinc est déjà assez coûteuse. La Table 1 montre l’augmentation moyenne, médiane, l’écart-type et l’augmentation maximale du nombre de variables et

2. Disponible sur Github : <https://github.com/ptal/turbo/tree/v1.2.8>

	Variables				Contraintes			
	moyenne	médiane	écart-type	max	moyenne	médiane	écart-type	max
FlatZinc	9.42x	1.86x	18.63x	111.62x	24.94x	2.95x	67.91x	486.87x
TCN	53.61x	7.97x	151.61x	1133.22x	76.68x	6.21x	265.88x	1837.19x
Prétraité	22.06x	4.76x	50.48x	344.62x	36.39x	4.33x	115.18x	746.09x

TABLE 1 – Augmentation de la taille des réseaux de contraintes par rapport aux réseaux de contraintes Choco sur 89 instances.

Solveur	TCN	Autre	Égal
Choco	22.5%	28.1%	49.4%
OR-Tools	11.2%	57.3%	31.5%
Choco-64	6.7%	71.9%	21.3%
OR-Tools-64	0%	80.9%	19.1%

FIGURE 1 – Comparaison entre deux solveurs de l'état de l'art et le réseau de contraintes ternaire prétraité avec point fixe par war psur 89 instances.

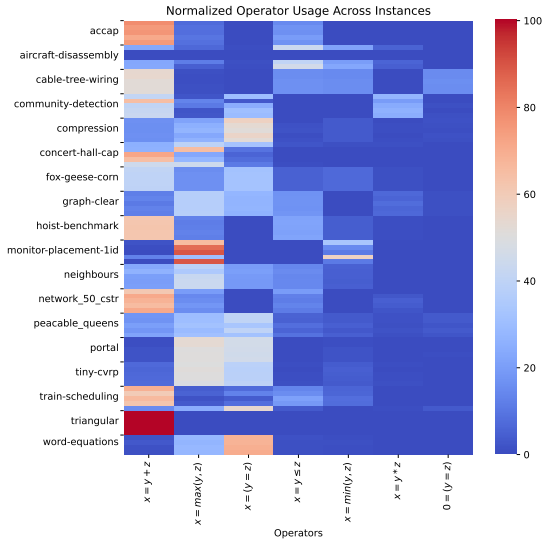


FIGURE 2 – Carte thermique des opérateurs utilisés dans les réseaux de contraintes ternaires prétraités sur 89 instances, regroupés par classes de problèmes.

de contraintes pour un modèle MiniZinc complètement décomposé (sans contraintes globales), après décomposition en TCN (Section 4) et après prétraitement (Section 5).

Sur 63 des 89 instances, la décomposition FlatZinc est inférieure à un ordre de grandeur par rapport au réseau de contraintes Choco, tant en nombre de variables qu'en nombre de contraintes. Après l'application de la décomposition TCN, nous augmentons encore de 5 fois le nombre de variables et de 3 fois le nombre de contraintes en moyenne. Heureusement, l'étape de prétraitement réduit cette augmentation à 2,5 fois pour les variables et 1,5 fois pour les contraintes en moyenne, bien qu'elle ne contienne que des contraintes ternaires. Le temps moyen de prétraitement est de 24,22s avec un écart-type de 96,91s. Le temps médian est de 0,91s et seulement 11 instances prennent plus de 10 secondes pour être prétraitées.

Sur la Figure 2, nous analysons l'utilisation des opérateurs à travers les instances. De toute évidence, la divergence de threads n'est plus la principale préoccupation puisque les instances n'utilisent pas une grande variété d'opérateurs. En particulier, il n'y a aucune instance avec des opérations de modulo et de division (ou elles sont simplifiées après le prétraitement). En raison du prétraitement, toutes les contraintes d'égalité non réifiées $1 = (x = y)$ sont supprimées. La contrainte $x = y \leq z$ n'existe que dans un contexte réifié et $0 = y \leq z$ (modélisant $y > z$) et

$1 = y \leq z$ ont complètement disparu.

Cela est dû à la décomposition FlatZinc et TCN, où $y \leq z$ est réécrit en $y - z \leq 0$ par la décomposition FlatZinc, puis réécrit en $y = x + z \wedge x \leq 0$, ce qui permet de gérer directement $\leq$ et $>$ dans le domaine des variables.

Les contraintes en TNF les plus utilisées sont l'addition, le maximum (qui est utilisé pour encoder la disjonction) et l'égalité réifiée. De manière intéressante, bien que présente dans 5 classes de problèmes, la décomposition de la contrainte globale *all-different* en $\mathcal{O}(n^2)$ contraintes de la forme $0 = (x = y)$ ne semble pas constituer un goulet d'étranglement.

Analyse de TCN pour la propagation sur GPU. Afin de valider notre décomposition TCN sur GPU, nous la comparons à une représentation traditionnelle des propagateurs basée sur les vues [37, 14], également exécutée sur GPU. Nous effectuons nos benchmarks sur un ensemble restreint de 20 instances, allant de petites à grandes tailles (au moins une pour chaque classe de problème), afin d'économiser des ressources.

Dans la Table 2, nous comparons l'efficacité brute en termes de nœuds par seconde et d'itérations de point fixe par seconde. Nous désactivons le prétraitement lors de la comparaison entre la propagation basée sur bytecode et celle basée sur les vues car nous ne disposons pas du même

	Basé sur Bytecode TCN Prétraité	Basé sur Bytecode Propagation	Basé sur Vues TCN Prétraité	Basé sur Vues Propagation
Nœuds par seconde	103036	79884	24276	14623
Itérations de point fixe par seconde	1429274	1379686	291438	88944
Itérations de point fixe par nœud	13.9	17.3	12	6.1
Mémoire des propagateurs (MB)	0.69	0.91	15.82	10.33
Mémoire des variables (KB)	286.5	397.3	286.5	76.6

TABLE 2 – Analyse de l’efficacité de la représentation en bytecode (prétraitée) de TCN, de la représentation basée sur les vues de TCN et de la représentation basée sur les vues de contraintes arbitraires. La moyenne est utilisée pour agréger les résultats. FP signifie point fixe.

algorithme de prétraitement pour les deux. La propagation basée sur bytecode explore environ 5 fois plus de nœuds par seconde en moyenne que la propagation basée sur les vues. Comme elle converge également près de 3 fois plus lentement, cela signifie que les propagateurs TCN pourraient bénéficier d’un tri plus intelligent pour accélérer la convergence, offrant ainsi la possibilité de tripler le nombre de nœuds explorés par seconde. La propagation basée sur les vues sur GPU bénéficie également de la représentation TCN, bien qu’elle soit clairement surpassée par la propagation basée sur bytecode.

De manière intéressante, bien que la décomposition TCN soit plus volumineuse, la représentation en bytecode est si compacte qu’elle utilise beaucoup moins de mémoire que les propagateurs basés sur les vues. Cependant, le stockage des variables est plus volumineux.

Analyse des algorithmes de point fixe. Nous comparons le point fixe par warp (WAC1) introduit dans la Section 3 avec la boucle de point fixe AC1. Globalement, WAC1 explore en moyenne 10% de nœuds en plus par rapport à AC1 et trouve une meilleure borne sur 12% des problèmes tout en étant égal sur les autres. De plus, en termes de vitesse de convergence—mesurée par le nombre de propagateurs appelés pour atteindre le point fixe—WAC1 est 30% plus rapide en moyenne et converge plus rapidement sur 83 des 89 instances. Par conséquent, il est raisonnable de conclure que WAC1 est une optimisation utile.

Plus important encore, cela montre qu’il existe des opportunités pour améliorer la planification des propagateurs afin d’accélérer davantage la convergence du plus grand point fixe. À ce stade, il n’est pas clair si une stratégie de planification dynamique serait efficace sur GPU en raison du surcoût lié à la planification des propagateurs en parallèle. Nous laissons cette question ouverte pour les recherches futures.

Comparaison avec Choco et OR-Tools. Dans cette dernière série d’expériences, nous comparons notre approche avec deux solveurs de l’état de l’art. Les résultats globaux sont illustrés dans la Figure 1. Malgré la simplicité de notre solveur, il n’est pas loin de Choco en mode séquentiel. Cela démontre que l’optimisation matérielle peut surpasser l’optimisation algorithmique telle que les contraintes globales et le redémarrage sur une certaine classe de problèmes. Le meilleur solveur OR-Tools ne nous dépasse pas complète-

ment, car nous trouvons de meilleures bornes sur 10 instances.

Pour être complets, nous présentons également la version parallèle de chaque solveur en utilisant 64 threads, bien que la comparaison soit difficile en raison des stratégies de parallélisation différentes. Choco et OR-Tools utilisent tous deux une parallélisation portfolio impliquant différentes stratégies de recherche, y compris des stratégies aléatoires. Nous sommes surpassés par ces deux solveurs. Pour une meilleure évaluation, nous devrions implémenter une recherche parallèle portfolio au lieu de EPS dans notre solveur GPU, en attribuant une stratégie de recherche par bloc. Nous laissons cela pour des travaux futurs.

7 Conclusion

Selon la thèse de Rich Sutton, tout ce dont nous avons besoin, c’est de plus de puissance de calcul et d’algorithmes évolutifs basés uniquement sur la recherche et l’apprentissage [39]. Dans cet article, nous proposons un algorithme de résolution évolutif et simple, accéléré sur GPU, basé sur la consistance des bornes et la recherche. Lorsqu’il est exécuté en parallèle sur un seul GPU, notre algorithme est compétitif avec le solveur de contraintes séquentiel Choco. Par conséquent, cette expérience soutient la thèse de Rich Sutton.

Cependant, nous devons réévaluer nos résultats en tenant compte d’un solveur hybride SAT-CP tel que OR-Tools. Dans ce cas, davantage de calculs ne suffisent pas à dépasser l’apprentissage de conflits. Le problème immédiat est que l’apprentissage de conflits est intrinsèquement un processus séquentiel, et difficilement parallélisable, bien qu’il existe quelques tentatives infructueuses [20]. La question est alors : peut-on se passer de l’apprentissage de conflits ?

Lors de la compétition XCSP3 2024, en comparant les solveurs basés sur SAT (par exemple, OR-Tools et Picat) avec ACE, un solveur purement basé sur la recherche, il a été montré qu’ACE est meilleur pour trouver de bonnes bornes d’objectif, tandis que les solveurs basés sur les conflits sont meilleurs pour prouver l’optimalité [5]. Par conséquent, il est possible que même sans apprentissage de conflits, un solveur de contraintes basé sur GPU puisse devenir compétitif avec les solveurs hybridés avec SAT.

Remerciements

Ce travail est supporté par le fond national de la recherche Luxembourgeois (FNR)—project COMOC, ref. C21/IS/16101289. Nous remercions également l’infrastructure Polonaise de calcul informatique haute performance PLGrid (HPC Center : ACK Cyfronet AGH) pour nous avoir fourni l’accès aux GPU H100 (computational grant no. PLG/2025/017986).

Références

- [1] Ahmed Abbas and Paul Swoboda. FastDOG : Fast Discrete Optimization on GPU. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 439–449, New Orleans, LA, USA, June 2022. IEEE.
- [2] David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O’Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. *Advances in Neural Information Processing Systems*, 34 :20243–20257, 2021.
- [3] Krzysztof R. Apt. The essence of constraint propagation. *Theoretical computer science*, 221(1-2) :179–210, 1999.
- [4] Alejandro Arbelaez and Philippe Codognet. A GPU Implementation of Parallel Constraint-Based Local Search. In *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 648–655, Torino, Italy, February 2014. IEEE.
- [5] Gilles Audemard, Christophe Lecoutre, and Emmanuel Lonca. Proceedings of the 2024 xcsp3 competition, 2024.
- [6] Nathan Bell and Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–11, Portland Oregon, November 2009. ACM.
- [7] Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers, and Jean-François Puget. Revising hull and box consistency. In *Proceedings of the 1999 International Conference on Logic Programming*, page 230–244, USA, 1999. Massachusetts Institute of Technology.
- [8] Frédéric Benhamou and William J. Older. Applying interval arithmetic to real, integer, and boolean constraints. *The Journal of Logic Programming*, 32(1) :1 – 24, 1997.
- [9] Frederic Boussemart, Christophe Lecoutre, Gilles Audemard, and Cédric Piette. XCSP3 : An Integrated Format for Benchmarking Combinatorial Constrained Problems. *arXiv e-prints*, November 2016.
- [10] Federico Campeotto, Alessandro Dal Palu, Agostino Dovier, Ferdinando Fioretto, and Enrico Pontelli. Exploring the use of GPUs in constraint solving. In *International Symposium on Practical Aspects of Declarative Languages*, pages 152–167. Springer, 2014.
- [11] Mats Carlsson, Greger Ottosson, and Björn Carlson. An open-ended finite domain constraint solver. *Programming Languages : Implementations, Logics, and Programs*, 1997.
- [12] Antonin Chambolle and Thomas Pock. A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging. *Journal of Mathematical Imaging and Vision*, 40(1) :120–145, May 2011.
- [13] Philippe Codognet and Daniel Diaz. Compiling constraints in clp(fd). *The Journal of Logic Programming*, 27(3) :185 – 226, 1996.
- [14] Marco Correia and Pedro Barahona. View-based propagation of decomposable constraints. *Constraints*, 18(4) :579–608, October 2013.
- [15] Agostino Dovier, Andrea Formisano, Enrico Pontelli, and Fabio Tardivo. {CUDA} : Set Constraints on GPUs. page 28, 2021.
- [16] Mokhtar Essaid, Lhassane Idoumghar, Julien Lepagnot, and Mathieu Brévilliers. GPU parallelization strategies for metaheuristics : a survey. *International Journal of Parallel, Emergent and Distributed Systems*, 34(5) :497–522, September 2019.
- [17] Ian P. Gent, Ian Miguel, Peter Nightingale, Ciaran McCreesh, Patrick Prosser, Neil CA Moore, and Chris Unsworth. A review of literature on parallel constraint solving. *Theory and Practice of Logic Programming*, 18(5-6) :725–758, 2018.
- [18] J. Gmys, M. Mezma, N. Melab, and D. Tuytens. A GPU-based Branch-and-Bound algorithm using Integer–Vector–Matrix data structure. *Parallel Computing*, 59 :119–139, November 2016.
- [19] Jan Gmys. Exactly Solving Hard Permutation Flowshop Scheduling Problems on Peta-Scale GPU-Accelerated Supercomputers. *INFORMS Journal on Computing*, 34(5) :2502–2522, September 2022.
- [20] Youssef Hamadi and Lakhdar Sais, editors. *Handbook of Parallel Constraint Reasoning*. Springer International Publishing, Cham, 2018.
- [21] Pieter Hijma, Stijn Heldens, Alessio Sclocco, Ben Van Werkhoven, and Henri E. Bal. Optimization Techniques for GPU Programming. *ACM Computing Surveys*, 55(11) :1–81, November 2023.
- [22] Beichen Huang, Ran Cheng, Zhuozhao Li, Yaochu Jin, and Kay Chen Tan. EvoX : A distributed GPU-accelerated framework for scalable evolutionary computation. *IEEE Transactions on Evolutionary Computation*, 2024. Publisher : IEEE.
- [23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional

- neural networks. *Advances in neural information processing systems*, 25, 2012.
- [24] Christophe Lecoutre. *Constraint networks : techniques and algorithms*. ISTE/John Wiley, Hoboken, NJ, 2009.
- [25] Alan K. Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1) :99–118, 1977.
- [26] Arnaud Malapert, Jean-Charles Régin, and Mohamed Rezgoui. Embarrassingly Parallel Search in Constraint Programming. *Journal of Artificial Intelligence Research*, 57 :421–464, November 2016.
- [27] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc : Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming—CP 2007*, pages 529–543. Springer, 2007.
- [28] Peter Nightingale, Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Patrick Spracklen. Automatically improving constraint models in Savile Row. *Artificial Intelligence*, 251 :35–61, October 2017.
- [29] NVIDIA Corporation. NVIDIA Grace Hopper Superchip Architecture. 2024.
- [30] NVIDIA Corporation. *CUDA Toolkit Documentation*, 2025. Accessed : 2025-03-04.
- [31] NVIDIA Corporation. *libcu++*, 2025. Accessed : 2025-03-04.
- [32] Laurent Perron and Frédéric Didier. Cp-sat, 2024.
- [33] Charles Prud’homme and Jean-Guillaume Fages. Choco-solver : A java library for constraint programming. *Journal of Open Source Software*, 7(78) :4708, 2022.
- [34] Andrea Rendl. *Effective compilation of constraint models*. PhD Thesis, University of St Andrews, 2010.
- [35] Christian Schulte. Comparing trailing and copying for constraint programming. In *Proceedings of the International Conference on Logic Programming*, pages 275–289. The MIT Press, 1999.
- [36] Christian Schulte and Peter J. Stuckey. Efficient Constraint Propagation Engines. *ACM Trans. Program. Lang. Syst.*, 31(1) :2 :1–2 :43, December 2008.
- [37] Christian Schulte and Guido Tack. View-based propagator derivation. *Constraints*, 18(1) :75–107, January 2013.
- [38] Boro Sofranac, Ambros Gleixner, and Sebastian Pokutta. Accelerating domain propagation : An efficient GPU-parallel algorithm over sparse matrices. *Parallel Computing*, 109 :102874, March 2022.
- [39] Richard Sutton. The bitter lesson. *Incomplete Ideas (blog)*, 13(1) :38, 2019.
- [40] Guido Tack. *Constraint Propagation – Models, Techniques, Implementation*. PhD thesis, Saarland University, 2009.
- [41] Pierre Talbot, Frédéric Pinel, and Pascal Bouvry. A Variant of Concurrent Constraint Programming on GPU. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3830–3839, Jun. 2022.
- [42] Yujin Tang, Yingtao Tian, and David Ha. Evojax : Hardware-accelerated neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 308–311, 2022.
- [43] Fabio Tardivo, Agostino Dovier, Andrea Formisano, Laurent Michel, and Enrico Pontelli. Constraint propagation on GPU : A case study for the AllDifferent constraint. *Journal of Logic and Computation*, 33(8) :1734–1752, December 2023.
- [44] Fabio Tardivo, Agostino Dovier, Andrea Formisano, Laurent Michel, and Enrico Pontelli. Constraint propagation on GPU : A case study for the cumulative constraint. *Constraints*, 29 :192–214, 2024. Publisher : Springer.
- [45] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.
- [46] Pascal Van Hentenryck, Vijay Saraswat, and Yves Deville. Design, implementation, and evaluation of the constraint language cc(FD). *The Journal of Logic Programming*, 37(1–3) :139–164, 1998.
- [47] Michael Vollmer, Sarah Spall, Buddhika Chamith, Laith Sakka, Chaitanya Koparkar, Milind Kulkarni, Sam Tobin-Hochstadt, and Ryan R. Newton. Compiling Tree Transforms to Operate on Packed Representations. pages 29 pages, 728133 bytes, 2017. Artwork Size : 29 pages, 728133 bytes ISBN : 9783959770354 Medium : application/pdf Publisher : [object Object].
- [48] Wen-mei W. Hwu and David B. Kirk and Izzat El Hajj. *Programming Massively Parallel Processors (Fourth Edition)*. Morgan Kaufmann, 2023.

Session 7b, mercredi 2 juillet, 11h25-12h00 : Tests & Vérifications

Tests à données aléatoires par mutations pour les systèmes de programmation par contraintes

Wout Vanroose¹, Ignace Bleux¹, Jo Devriendt¹, Dimos Tsouros¹, Hélène Verhaeghe^{1,2}, Tias Guns¹

¹ KULEuven, Louvain, Belgique

² UCLouvain, Louvain-la-Neuve, Belgique

wout.vanroose@kuleuven.be, ignace.bleux@kuleuven.be, jo.devriendt@kuleuven.be,
dimos.tsouros@kuleuven.be, helene.verhaeghe@uclouvain.be, tias.guns@kuleuven.be

Résumé

Les langages de modélisation de programmation par contraintes (PPC), tels que MiniZinc, Essence et CPMpy, jouent un rôle dans l'accessibilité des techniques de PPC aux non-experts. Les langages de modélisation, comme les solveurs, sont des pièces complexes de logiciels pouvant contenir des bogues, réduisant leur utilité. Les tests à données aléatoires par mutations sont une manière de tester des systèmes complexes en mutant des entrées et en vérifiant la préservation de propriétés dans la sortie mutée. Ce papier est un résumé de l'article "Mutational Fuzz Testing for Constraint Modeling Systems"[3] publié à CP2024.

Mots-clés

Tests à données aléatoires par mutations, langage de modélisation de problèmes PPC, bogue

Abstract

Constraint programming (CP) modeling languages, like MiniZinc, Essence and CPMpy, play a crucial role in making CP technology accessible to non-experts. Both solver-independent modeling frameworks and solvers themselves are complex pieces of software that can contain bugs, which undermines their usefulness. Mutational fuzz testing is a way to test complex systems by stochastically mutating input and verifying preserved properties of the mutated output. This paper is a summary of the article "Mutational Fuzz Testing for Constraint Modeling Systems"[3] published at CP2024.

Keywords

Fuzz testing, Constraint modeling language, bugs.

1 Introduction

La résolution sous contraintes est une technique de raisonnement d'IA déclarative qui est utilisée pour une variété d'applications allant de la planification de lignes de productions à l'automatisation de programmes informatiques en passant par des applications aérospatiales. Toutes ces applications requièrent que les solveurs de contraintes donnent des solutions correctes et fiables vis-à-vis du modèle.

Pour utiliser la puissance des solveurs modernes, il est commun pour les utilisateurs d'écrire les spécifications de leurs problèmes dans un langage de modélisation de contraintes haut niveau tel que MiniZinc, XCSP, Essence ou CPMpy[1]. Ces langages de modélisation jouent un rôle fondamental pour l'adoption à large échelle des technologies de programmation par contraintes (PPC) à travers de nombreux domaines grâce à leur formalisme haut niveau, leur expressivité et les méthodes de définition de problèmes intuitives pour les utilisateurs. Ils offrent une abstraction des détails de l'encodage des contraintes de haut niveau dans les contraintes spécifiques prises en charge par un solveur, permettant aux utilisateurs de se concentrer sur le problème en cours plutôt que sur les spécificités des solveurs.

Les systèmes de modélisation reformulent ensuite les contraintes utilisateur de haut niveau en des expressions spécifiques aux solveurs telles que des clauses, des contraintes linéaires ou des contraintes globales non imbriquées. Pour cette étape, le code de base des systèmes de modélisation inclut un certain nombre de reformulations et d'algorithmes d'encodage. Ces systèmes sont également complexifiés par la présence d'optimisations de modèles tels que les systèmes d'élimination de sous-expression commune.

Comme de nombreux logiciels complexes, les systèmes de modélisation et les solveurs de contraintes peuvent contenir des bogues. Dans le cas des systèmes de modélisation, ceux-ci peuvent avoir plusieurs types de comportements indésirables allant de l'interruption de fonctionnement du système à la génération de fausses solutions ou du retrait de solutions valides (ex : retrait de l'optimal). Ces situations peuvent avoir un impact important sur l'utilisateur et son application. De plus, cela peut diminuer la confiance des utilisateurs envers un solveur.

Pour ce faire, nous proposons le logiciel cadre *Hurricane*, basé sur les techniques de vérification automatique par données aléatoires par mutations[2] pour vérifier les systèmes de modélisation de programmation par contraintes.

2 Tests à données aléatoires par mutations

Notre système de tests à données aléatoires par mutations fonctionne de la manière suivante. Un ensemble de m problèmes de satisfaction ou d'optimisation de contraintes satisfaisables est sélectionné. A chaque itération de l'algorithme, un modèle est sélectionné de manière aléatoire et un certain nombre de mutations sont appliquées aux contraintes de ce modèle. Une mutation est une fonction qui prend en entrée un ensemble de contraintes et en sortie donne un autre ensemble de contraintes. Ces nouvelles contraintes sont ajoutées au modèle. Après application des mutations, nous vérifions si certaines propriétés (dépendantes du type de mutation) sont conservées. Si la vérification échoue, l'algorithme a découvert un bogue dans le système et l'enregistre pour analyse par l'utilisateur.

2.1 Ensemble de modèle de départ

Avoir des modèles simples permet une plus grande facilité à identifier la source en cas d'erreur. Pour également diminuer la complexité de création de cet ensemble et s'assurer que tous les types de contraintes supportées sont utilisées, la meilleure piste est de partir des modèles utilisés dans les tests unitaires. Ces modèles sont généralement petits, compréhensibles et contiennent un nombre restreint de contraintes et variables à la fois.

2.2 Mutations

Les mutations sont de plusieurs types, classées en fonction des propriétés qu'elles respectent mais également du type de modifications appliquées sur les contraintes.

2.2.1 Reformulations

Les systèmes de modélisation possèdent généralement plusieurs fonctions de reformulations afin de transformer les modèles définis par les utilisateurs vers les solveurs. Ces reformulations incluent entre autre la transformation en contraintes linéaires, la décomposition de contraintes globales ou la simplification de termes. Ces transformations peuvent servir directement de mutation.

2.2.2 Mutations haut niveau

Les mutations haut niveau sont des mutations basées sur la logique entre deux expressions booléennes. Par exemple, si le modèle contient deux contraintes A et B , la contrainte $A \wedge B$ est validée pour toutes les solutions du modèle initial, de même que pour $A \vee B$. Ajoutée au modèle, les mutations de ce type ne change pas l'ensemble de solutions acceptées.

2.2.3 Mutation de sous-expressions : fusion sémantique

L'idée de ce type de mutation est de fusionner deux expressions et créer une variable auxiliaire pour celle-ci. Par exemple, si on a les variables x_1 et x_2 , on peut définir $y = x_1 + x_2$. Cette nouvelle expression est alors utilisée pour substituer les anciennes expressions là où elles sont utilisées. Dans le cas de l'exemple, x_1 sera substitué par $y - x_2$ et x_2 sera substitué par $y - x_1$.

2.2.4 Mutation de sous-expressions : comparaisons équivalentes

Ce type de mutation sélectionne une comparaison de manière aléatoire et applique une multiplication par une constante (positive) de chaque coté de la comparaison.

2.3 Méthodes de vérifications

Une fois le modèle muté obtenu, il faut vérifier si un bogue a été potentiellement introduit. Pour cela, une méthode de vérification est sélectionnée. Certaines sont plus coûteuse, mais permettent plus de sécurité quant à la découverte d'un bogue sur l'exemple.

Ces méthodes de vérifications sont les suivantes :

- vérifier toutes les solutions : comparer l'ensemble de solutions du modèle non muté avec l'ensemble de solutions du modèle muté, si ceux-ci sont différents, il existe un bogue
- compter le nombre de solutions : comparer le nombre de solutions des deux modèles, si le nombre diffère, il y a un bogue, un nombre égal n'impliquant pas forcément une absence de bogue
- vérifier une solution spécifique : sélectionner une solution du modèle non muté et vérifier qu'elle est toujours acceptée par le modèle muté
- vérifier la satisfaction : vérifier que le problème est toujours satisfaisable
- vérifier l'optimum : vérifier que l'optimum est toujours le même

3 Discussion sur les résultats et conclusion

Nous avons fait tourner *Hurricane* sur le système de modélisation CPMpy et avons découvert un nombre significatif de bogues, appartenant à plusieurs catégories et non détectés par les tests unitaires. Certains causaient des arrêts du système, tandis que d'autres modifiaient l'ensemble de solutions. Notre utilisation a montré également une augmentation de la couverture de test du système. Notre méthode est également particulièrement bien conceptualisée pour de l'intégration continue, et permet de découvrir de nouveaux bogues au fur et à mesure du développement. Cependant, bien que notre méthode soit efficace, l'une des principales difficultés est la potentielle découverte d'un même bogue via plusieurs modèles mutés. Il est en effet difficile, sans analyse humaine, de détecter si la source du bogue est la même.

Notre logiciel cadre est également très modulable et peut évoluer en fonction des besoins du logiciel à tester. Il est en effet facile d'ajouter de nouveaux modèles de base, ainsi que de nouvelles mutations. Pour les systèmes de modélisation intégrant plusieurs solveurs cibles, une nouvelle méthode de vérification simple peut être de résoudre le modèle muté avec plusieurs solveurs et voir si ceux-ci s'accordent sur la vérification. Nous pensons donc qu'il peut être utile pour tester tout système de modélisation.

Remerciements

Ce projet a reçu du financement du Conseil Européen de la Recherche (ERC) sous le programme de recherche et d'innovation Horizon 2020 (No. 101002802, CHATOpt).

Références

- [1] Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.
- [2] Muhammad Numair Mansur, Maria Christakis, Valentin Wüstholtz, and Fuyuan Zhang. Detecting critical bugs in SMT solvers using blackbox mutational fuzzing. In Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann, editors, *ESEC/FSE '20 : 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, pages 701–712. ACM, 2020.
- [3] Wout Vanroose, Ignace Bleux, et al. Mutational fuzz testing for constraint modeling systems. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307, pages 29–1. Schloss Dagstuhl–Leibniz-Zentrum, 2024.

Domaines formellement vérifiés

Catherine Dubois

ENSIIE, INRIA, Université Paris-Saclay, LMF, France

catherine.dubois@ensiie.fr

Résumé

Cet article s'intéresse à la correction de la représentation des domaines des variables dans un solveur de contraintes. Nous proposons d'utiliser la vérification déductive, et plus précisément l'outil Why3, afin de fournir des implémentations correctes des domaines. L'article résume les travaux réalisés par l'autrice, concernant la vérification formelle des sparse sets pour représenter les domaines finis entiers. La contribution nouvelle concerne le développement d'une implémentation formellement vérifiée des domaines ensemblistes, basée, elle aussi, sur les sparse sets.

Mots-clés

vérification formelle, domaines entiers, domaines ensemblistes, sparse sets, Why3

1 Introduction

Développer un solveur de contraintes requiert entre autres de choisir une représentation efficace et correcte pour les domaines des variables. Différentes structures de données sont utilisées dans les solveurs existants : séquences ou arbres d'intervalles, bit-vecteurs, ensembles épars (*sparse sets* en anglais et dans la suite), etc. Nous proposons de développer de telles représentations, en utilisant un assistant à la preuve puis d'en extraire un code exécutable. L'assistant à la preuve nous permet d'implémenter, spécifier et prouver que l'implémentation est correcte par rapport à la spécification. À l'aide de Coq, nous avons formalisé les séquences d'intervalles pour représenter les domaines des variables entières [6]. Ici, nous nous concentrons sur les *sparse sets* et les domaines ensemblistes représentés avec une variante des *sparse sets*, à l'aide de l'assistant à la preuve Why3. La contribution nouvelle est la formalisation en Why3 des domaines ensemblistes. La première contribution a fait l'objet de la publication [2] que nous résumons dans l'article.

Nous mettons l'accent sur les spécifications des structures de données, en particulier leurs invariants ainsi que sur les spécifications des opérations. Nous démontrons en particulier que les opérations établissent ou préservent les invariants. Ces développements, dont le code complet est disponible, ont permis d'extraire du code exécutable en OCaml. Nous avons choisi l'outil de vérification déductive Why3 car il y est aisé de formaliser et prouver des algorithmes manipulant des données mutables, puis d'en extraire du code exécutable efficace. Il est plus difficile de représenter des structures de données mutables en Coq, les fonctionnalités

impératives, comme par exemple la modification en place de tableaux, n'y étant pas directement disponibles.

Une approche alternative consiste à prouver la correction des implémentations existantes. Cette approche est impossible pour le code d'un solveur dans son ensemble car celui-ci est trop complexe. Elle pourrait cependant être possible pour les modules relatifs aux domaines (e.g. dans FaCiLe [6]). Cependant, en lisant, par exemple, le code de MiniCP ou OscaR, on se rend compte que de nombreux aspects sont imbriqués et qu'il est difficile d'isoler les seuls éléments des structures de données d'étude.

2 Présentation rapide de Why3

Why3 [3] est une plateforme de vérification déductive de programmes qui fournit un langage de spécification et de programmation, appelé WhyML. Ce dernier permet à l'utilisateur d'écrire des programmes fonctionnels ou impératifs comportant du polymorphisme, des types et prédicats inductifs, du filtrage, des exceptions, des références, des tableaux, etc. Les programmes sont spécifiés à l'aide de contrats contenant des pré- et post-conditions et complétés éventuellement par des invariants de boucle. La logique sous-jacente est une extension de la logique du premier ordre. À partir d'un programme annoté par sa spécification, Why3 génère des conditions de vérification (VC), formules logiques obtenues par un calcul de plus faible pré-condition et dont la preuve assure la correction du programme par rapport à sa spécification. Pour démontrer les VCs, Why3 s'appuie sur des prouveurs automatiques standards (nous utilisons ici Alt-Ergo, CVC4 et Z3) ainsi que des assistants de preuve interactifs tels que Coq ou Isabelle (que nous n'avons pas utilisés ici). Des tactiques de preuve sont également fournies, pour guider la preuve ou comprendre les manques ou erreurs au cours du développement.

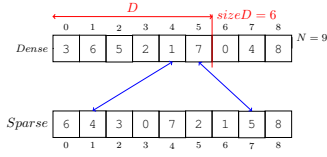
Il existe de nombreuses bibliothèques qui peuvent être utilisées, pour l'arithmétique des entiers, les listes polymorphes, les ensembles finis, les fonctions partielles, etc.

À partir des programmes WhyML vérifiés, des programmes OCaml (et récemment des programmes C) corrects par construction peuvent être automatiquement extraits.

3 Sparse sets vérifiés

Un *sparse set* permet de représenter le domaine d'une variable entière, inclus dans un intervalle $[0, N - 1]$ ¹, où

1. Ceci est un peu restrictif mais il est facile de retrouver le cadre plus général en utilisant un *offset* comme cela est fait dans OscaR par exemple.


 FIGURE 1 – Un *sparse set*

N est un entier naturel non nul. Cette structure de données utilise deux tableaux de longueur N appelés *Dense* et *Sparse*, et un entier naturel *sizeD*. Les éléments du domaine, *setD*, sont les entiers présents dans le sous-tableau *Dense*[0..*sizeD* - 1]. Le tableau *Sparse* associe toute valeur v de $[0, N - 1]$ à sa position i dans *Dense*. Les invariants de la structure de données sont les suivants :

$$\text{setD} \subseteq [0..N - 1] \wedge \text{setD} = \text{Dense}[0..\text{sizeD} - 1] \quad (P_1)$$

$$\text{Sparse}[v] = i \iff \text{Dense}[i] = v, \text{ for all } i, v \quad (P_2)$$

Un exemple est donné à la figure 1. Les flèches bleues y illustrent l'invariant P_2 : *Dense*[1] = 4, *Sparse*[4] = 1.

Cette représentation est particulièrement efficace pour le test d'appartenance et les opérations *remove* (qui retire un élément du domaine) et *bind* (qui lie une variable à une valeur), qui sont en temps constant. Tester l'appartenance d'un élément v au domaine consiste à évaluer l'expression $v < \text{sizeD}$, retirer v revient à échanger, dans *Dense*, v avec *Dense*[$N - 1$], et de faire les modifications correspondantes dans *Sparse*, réduire le domaine à $\{v\}$ consiste à échanger v avec *Dense*[0] et mettre à jour *Sparse*.

La formalisation en WhyML suit très fidèlement cette présentation. Un extrait en est donné dans le listing 1. Le type des *sparse sets*, *tsparse*, est défini ici comme un type enregistrement comprenant, outre les champs *n*, *dense* et *sparse*, un champ *setD* dit fantôme. Il aura pour valeur le domaine courant et n'est là que pour les besoins de la spécification et faciliter la preuve. Lors de l'extraction du code OCaml, il sera effacé. Cette valeur fantôme, un ensemble mathématique de type polymorphe *fset int*, est mise à jour dans les différentes fonctions qui modifient un *sparse set*. Dans la suite, nous écrivons ce code fantôme en italiques et utilisons les notations mathématiques usuelles pour dénoter les opérations ensemblistes mathématiques. Le type *tsparse* est accompagné d'un invariant qui reprend les deux propriétés P_1 et P_2 énoncées plus haut. *dom_ran dense n* spécifie que n est positif ou nul, que le tableau *dense* est de longueur n et que ses éléments sont compris entre 0 et $n - 1$. L'invariant est vérifié aux limites des appels de fonction : chaque fois qu'un *sparse set* est créé ou modifié, une VC correspondant resp. à l'établissement ou la préservation de l'invariant est générée.

Le contrat associé à la fonction *remove* spécifie que la valeur v à retirer doit appartenir au domaine² et que son effet est de retirer cette valeur. Pour exprimer cette propriété, le modèle abstrait, soit la variable fantôme *setD*, est utilisé. L'opérateur *old*, utilisé dans une post-condition, fait réfé-

rence à la valeur d'un terme au point d'appel de la fonction. Une condition de vérification correspondant à la satisfaction de cette post-condition est générée.

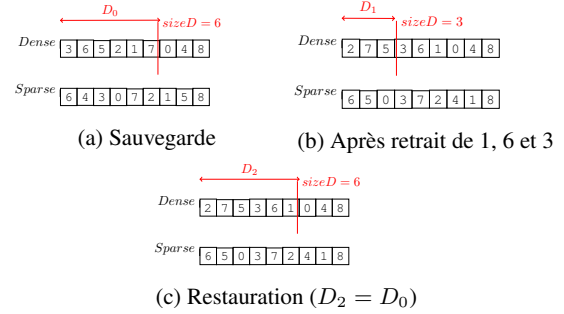
```
type tsparse = { n : int;
  mutable dense: array int; mutable sparse: array int;
  mutable sizeD: int;
  mutable ghost setD : fset int; }

invariant {
  dom_ran dense n && dom_ran sparse n && 0 <= sizeD <= n &&
  setD ⊆ [0, n-1] &&
  (forall x. 0 <= x < n -> (sparse[x] < sizeD <-> x ∈ setD)) &&
  (forall i. 0 <= i < n -> sparse[dense[i]] = i) }

let remove (v : int) (a : tsparse)
requires 0 <= v < a.n && v ∈ a.setD
ensures {a.setD == (old a.setD) - {v}}
=
  swap_arrays a.dense a.sparse a.n a.sparse[v] (a.sizeD - 1);
  a.sizeD <- a.sizeD - 1;
  a.setD <- a.setD - {v}
```

 Listing 1 – *Sparse set* en WhyML (extrait)

Un *sparse set* est également peu coûteux à restaurer dans le cadre d'une recherche de solution utilisant le *backtracking*. Il suffit en effet de sauvegarder et restaurer la limite *sizeD*. Nous retrouvons le même ensemble mathématique, même si les deux tableaux sont différents. Cette situation est illustrée par un exemple dans la figure 2.


 FIGURE 2 – Restaurer un *sparse set*

Nous avons implémenté une fonction *undo* qui prend en paramètre la valeur de *sizeD* à restaurer. Si le code de la fonction est extrêmement simple, sa spécification est plus délicate et demande de revoir la définition du type *tsparse* ainsi que la plupart des opérations. Nous introduisons pour cela, dans *tsparse*, une nouvelle variable fantôme, *states*, qui stocke les différents états du domaine, à savoir les valeurs successives de l'ensemble *setD*. Elle est définie comme une fonction partielle qui associe à p le modèle de cardinal p ³. Ainsi *undo* (p) consiste à revenir à la situation antérieure où le modèle *setD* était *states* (p). La variable *states* est soumise à l'invariant spécifié par le prédicat *inv_states* (cf listing 2), qui est ajouté comme invariant du type *tsparse*. La dernière propriété fait le lien entre *states* et *sparse* : si *states*(i) = s alors s = *dense*[0.. i] (et donc tous les éléments de s ont une valeur dans *sparse* strictement inférieure à i).

```
predicate inv_states states setD n sparse =
-- le domaine de définition est inclus dans [|setD|, n]
(dom states) ⊆ [|setD|, n] &&
```

3. le cardinal de *setD* est exactement *sizeD*.

2. Cette pré-condition est ajoutée ici pour simplifier la présentation. Le code vérifié et extrait ne fait pas cette hypothèse et teste l'appartenance de v au domaine.

```

-- l'état courant y est enregistré
states (|setD|) = setD &&
-- chaque état stocké est bien formé et contient setD
forall i, s. states(i) = s ->
  s ⊆ [0, n-1] && |setD| = i && setD ⊆ s
-- lien avec sparse
(forall i, s. 0 <= i <= n -> states(i) = s ->
  (forall x. 0 <= x <= n -> (sparse[x] < i <-> x ∈ s)))

let undo (a : tsparse) (p : int) : unit
requires {a.sizeD < p <= a.n && p ∈ dom(a.states)}
ensures {exists s. (old a).states(p) = s && a.setD == s}
ensures {forall i, v. p < i <= a.n ->
  (a.states(i) = v <-> (old a).states(i) = v)}
=
let ghost (v : fset int) = a.states(p) in
a.setD <- v ; a.states <- [0, p-1] <# a.states ;
a.sizeD <- p

```

Listing 2 – Invariant de `states` et `undo` en WhyML

La fonction `undo` (cf listing 2) a une large part "fantôme" qui consiste à mettre à jour `setD` et `states`. L'opérateur $\Leftarrow$ permet de supprimer les potentiels éléments de l'intervalle $[0, p-1]$ du domaine de `states`. Nous assurons ainsi que les états stockés sont antérieurs à celui caractérisé par `p`.

La fonction `undo` a une pré-condition contraignant son paramètre à être dans le domaine de définition de `states`. Si la fonction extraite est utilisée sans vérifier cette pré-condition, l'invariant pourrait être violé. Une version défensive, vérifiant dans le corps de `undo` que le paramètre `p` est bien un "cardinal rencontré précédemment", a également été développée et vérifiée, impliquant d'utiliser, pour l'ensemble de la formalisation, un entier réversible pour la limite `sizeD`. On dispose ainsi non seulement de la valeur courante de `sizeD` mais aussi de ses valeurs antérieures. Le type `rint` d'un entier réversible est donné ci-dessous.

```

type rint = {mutable value : int;
             mutable back : list int;}
invariant {sorted back && no_duplicates back &&
  forall x. x ∈ back -> value < x}

```

Ainsi, dans la fonction `remove`, la valeur courante de l'entier réversible `sizeD` est poussée dans la liste `sizeD.back` avant d'être décrémentée.

Pour assurer la cohérence entre `sizeD` et les états stockés dans `states`, nous ajoutons la propriété suivante dans l'invariant du type `tsparse` :

```

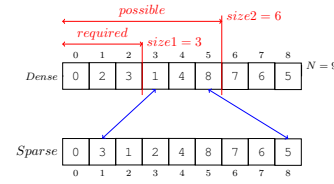
forall x. (x = sizeD.value || x ∈ sizeD.back) <->
  x ∈ dom states

```

Les preuves des VCs sont automatiques, moyennant l'ajout de quelques lemmes, eux aussi prouvés automatiquement. Pour plus de détails, se reporter au code en ligne (https://gitlab.com/cdubois/why3_sparsesets) et à [2].

4 Sparse subsets vérifiés

Cette section présente la formalisation en Why3 des domaines des variables qui modélisent des ensembles d'entiers, plus exactement des sous-ensembles d'un intervalle $[0, N - 1]$. Le domaine d'une variable X est alors représenté par un intervalle d'ensembles, dont la borne inférieure, appelée *required*, contient les éléments qui doivent être dans l'ensemble X , et la borne supérieure, appelée *possible*, contient les éléments qui peuvent apparaître dans l'ensemble X [4]. Ainsi le domaine de X est l'ensemble

FIGURE 3 – Un *sparse subset*

de tous les sous-ensembles de $[0, N - 1]$ qui contiennent *required* et sont contenus dans *possible* : $\{s \subseteq [0, N - 1] \mid \text{required} \subseteq s \subseteq \text{possible}\}$. Par exemple, l'intervalle d'ensembles $\{\{1, 3\}, \{1, 2, 3, 4\}\}$ représente le domaine ensembliste $\{\{1, 3\}, \{1, 2, 3\}, \{1, 3, 4\}, \{1, 2, 3, 4\}\}$.

La représentation concrète vérifiée réutilise et adapte la structure de données de *sparse set*, comme cela est proposé par Le Clément de Saint-Marcq et al. dans [5] et implémenté, par exemple, dans le solveur OslaR. Nous appelons cette structure de données un *sparse subset*. L'adaptation consiste en la manipulation de deux limites, *size₁* et *size₂*, au lieu d'une seule. Le sous-tableau `dense[0..size1 - 1]` contient les éléments requis (c'est-à-dire la borne inférieure de l'intervalle d'ensembles), le sous-tableau `dense[0..size2 - 1]` contient les éléments possibles (c'est-à-dire la borne supérieure de l'intervalle d'ensembles), le reste du tableau `dense` contient, quant à lui, les éléments qui ont été exclus du domaine au cours des filtrages successifs. Un exemple est présenté à la figure 3, il représente l'intervalle d'ensembles $\{\{0, 2, 3\}, \{0, 1, 2, 3, 4, 8\}\} \subseteq \mathcal{P}([0, 8])$.

La spécification d'un *sparse subset* est présentée en WhyML dans le listing 3. Nous retrouvons la taille `n`, les deux tableaux `dense` et `sparse`, les deux limites rassemblées en un couple "réversible", `size12` (de type `rpairs`), ceux-ci constituant la partie calculatoire, ainsi que les variables fantômes `setD`, `required` et `possible` qui servent à la spécification. Comme précédemment, cette structure de données est réversible : pour restaurer un *sparse subset*, il suffit de restaurer les valeurs des deux limites, donc la valeur du couple `size12`. Nous introduisons donc également une variable fantôme `states`, qui stocke les états successifs de la structure de données, c'est-à-dire ici un couple composé des deux ensembles `required` et `possible`. Elle est, de la même façon que pour les *sparse sets*, une fonction partielle définie sur des couples d'entiers. Les propriétés invariantes sont similaires à celles pour les *sparse sets*. Nous retrouvons, dans l'ordre, les propriétés qui décrivent le modèle abstrait, la propriété P_2 qui impose à `dense` et `sparse` d'être inverses l'un de l'autre, les propriétés qui font le lien entre les ensembles abstraits du modèle et les bornes *size₁* et *size₂* (`get1` et `get2` sont resp. la première et la deuxième projection des couples), la propriété spécifiant `states` et enfin la propriété assurant la cohérence entre `states` et le couple réversible des deux bornes. Le prédicat `inv_states_set states`, non développé ici, est similaire à celui du listing 2, il repose sur la définition d'un ordre sur les couples qui reflète le fait qu'au cours des filtrages successifs, l'ensemble `required` grossit (et donc *size₁* augmente) alors que `possible` rétrécit (et donc *size₂* décroît). Le champ `back` d'un couple réversible

d'entiers forme une liste triée selon cet ordre.

```

type tsparsesubset = {
  n : int;
  mutable dense: array(int); mutable sparse: array(int);
  mutable size12: rpair;
  ghost mutable setD: fset (fset int);
  ghost mutable required: fset int;
  ghost mutable possible: fset int;
  ghost mutable states: fmap (int,int) (fset int, fset int) }
invariant {
  possible  $\subseteq$  (interval 0 n) &&
  required  $\subseteq$  possible &&
  forall s. s  $\in$  setD  $\leftrightarrow$  (required  $\subseteq$  s  $\subseteq$  possible) &&
  dom_ran dense n && dom_ran sparse n &&
  (forall i. 0<=i<n  $\rightarrow$  sparse[dense[i]]=i) &&
  0<=get1(size12.value)<= n &&
  get1(size12.value)<=get2(size12.value) <= n &&
  (forall x. 0<=x<n  $\rightarrow$ 
    (sparse[x]<get1(size12.value)  $\leftrightarrow$  x  $\in$  required)) &&
  (forall x. 0<=x<n  $\rightarrow$ 
    (sparse[x]<get2(size12.value)  $\leftrightarrow$  x  $\in$  possible)) &&
  inv_states_set states required possible size12 sparse &&
  forall x. (x = sizeD.value || x  $\in$  sizeD.back)  $\leftrightarrow$ 
    x  $\in$  dom states }

```

Listing 3 – Type `tsparsesubset` en WhyML

Nous avons spécifié, implémenté et vérifié les opérations `create`, `full_interval`, `excludes_val`, `requires_val` (cf listing 4), `excludesAll`, `requiresAll`..., inspirées de celles développées dans *Oscar* ainsi qu'une opération `undo`. Les opérations `excludes`, `requires_val`, `excludesAll`, `requiresAll`, sont en temps constant et consistent en des échanges et des modifications des bornes. L'opération `undo` prend cette fois en paramètre deux entiers censés décrire une "situation antérieure", son code WhyML consiste à mettre à jour `size12` par le couple formé des deux paramètres, ainsi que les variables fantômes (exactement comme dans le cas des *sparse sets*).

```

--La fonction impose v parmi les valeurs requises - Elle
lève une exception si v est déjà exclue.
let requires_val (v : int) (a: tsparsesubset) : unit
requires {0<=v<a.n}
ensures {a.required == (old a.required) U {v}}
ensures {forall x. x  $\in$  a.setD  $\leftrightarrow$ 
  exists y. y  $\in$  (old a).setD && x == y U {v}}
raises {ImpossibleOp  $\rightarrow$  v  $\notin$  a.possible} && a = old a
=
let i = a.sparse[v] in
let (size1, size2) = a.size12.value in
if i < size1 then
else
  if i >= size2 then raise ImpossibleOp
  else
    begin
      swap_two_arrays a.dense a.sparse a.n a.sparse[v] size1;
      set (size1+1, size2) a.size12;
      a.required <- a.required U {v};
      a.setD <- addSet4 v a.setD;
      a.states <- a.states U
        {(a.size12, (a.required, a.possible))}
    end
end

```

Listing 4 – Type `tsparse` en WhyML

Afin d'extraire un code OCaml exécutable efficace, travaillant sur des entiers machine bornés et non des entiers mathématiques⁵, il reste une dernière étape. L'implémentation présentée dans cette section est modifiée pour utiliser des entiers machine, et ce, sans difficulté : remplacement

4. `addSet` est une fonction logique qui ajoute son premier paramètre dans chacun des ensembles de son deuxième paramètre.

5. le type `int` de WhyML désigne le type des entiers mathématiques

du type `int` par le type `int63` de la bibliothèque standard de Why3 et insertion de quelques conversions dans les parties logiques qui continuent de manipuler des entiers mathématiques. Une modification a dû toutefois être introduite dans les spécifications pour ne pas occasionner d'*overflow* dans l'opération `size` qui calcule la hauteur du treillis des sous-ensembles sous-jacent. Ainsi `a.n` est contraint à être strictement inférieur à `max_int`. Les preuves des VCs demeurent automatiques. Le code complet est disponible à l'adresse suivante : https://gitlab.com/cdubois/why3_sparsesubsets.

5 Conclusion

Le travail présenté ci-dessus a permis de mettre à disposition une bibliothèque d'implémentations formellement vérifiées pour représenter des domaines de variables dans un solveur de contraintes.

Le développement réalisé en Why3 pour les domaines ensemblistes a bénéficié grandement de celui réalisé au préalable concernant la représentation des *sparse sets*. Il est plus complexe dans la mesure où l'on est amené à manipuler un couple d'entiers (`size12`) au lieu d'un entier unique (`sizeD`) et deux ensembles (`required` et `possible`) au lieu d'un seul (`setD`) mais les spécifications sont très similaires. Il est sans doute possible de réutiliser une partie du code WhyML en généralisant certaines propriétés, ce qui n'a pas été fait ici. Nous envisageons également d'explorer en Why3 la représentation basée sur les *sparse bitsets* [1].

Références

- [1] J. Demeulenaere, R. Hartert, C. Lecoutre, G. Perez, L. Perron, J. Régim, and P. Schaus. Compact-table : Efficiently filtering table constraints with reversible sparse bit-sets. In *Principles and Practice of Constraint Programming (CP 2016)*, Toulouse, France, volume 9892 of *LNCS*, pages 207–223. Springer.
- [2] C. Dubois. Deductive Verification of Sparse Sets in Why3. In *Verified Software. Theories, Tools and Experiments, Prague, Czech Republic (VSTTE 2024)*, volume 15525 of *LNCS*, pages 28–46. Springer.
- [3] J.-C. Filliâtre and A. Paskevich. Why3 - where programs meet provers. In *European Symposium on Programming (ESOP 2013)*, Rome, Italy, volume 7792 of *LNCS*, pages 125–128. Springer.
- [4] C. Gervet. Conjunto : Constraint propagation over set constraints with finite set domain variables. In *International Conference on Logic Programming, Santa Margherita Ligure, Italy*. MIT Press, 1994.
- [5] V. Le Clément de Saint-Marcq, P. Schaus, C. Solnon, and C. Lecoutre. Sparse-Sets for Domain Implementation. In *Techniques for Implementing Constraint programming Systems (TRICS)*, pages 1–10, 2013.
- [6] A. Ledein and C. Dubois. Facile en coq : vérification formelle des listes d'intervalles. In *Journées Francophones des Langages Applicatifs (JFLA 2019)*, Gruissan, France.

Session 8, mercredi 2 juillet, 15h30-16h20 : Applications

Gestion des ressources de communication pour satellites dans une fédération de constellations d'observation de la Terre*

Hénoïk Willot¹, Jean-Loup Farges¹, Gauthier Picard¹, Philippe Pavero²

¹ DTIS, ONERA, Université de Toulouse

² Airbus Defence and Space

Résumé

Cet article aborde le problème nouveau, découlant du concept novateur de fédération de missions dans le domaine de l'observation de la Terre grâce au développement de réseaux de stations terrestres, rencontré par le système de communication et de gestion des ressources des satellites (SCRMS). Ce problème décisionnel est celui de la sélection des contacts, sous contraintes de satisfaction des besoins de communication des satellites, en minimisant le coût et les brouillages. Nous avons développé plusieurs méthodes de résolution : un programme linéaire, des schémas de recherche incomplète et des schémas d'allocation gloutonne.

Mots-clés

Observation de la Terre, ressources de communication, GSaaS, programmation linéaire en nombres entiers, recherche incomplète.

1 Introduction

Traditionnellement, les missions spatiales consacrées à l'observation de la Terre disposent de leurs propres ressources. Chaque mission possède sa constellation de satellites, ses stations au sol et son centre de contrôle. Les stations au sol sont dédiées à la communication avec les satellites et permettent de charger des plans d'observation et de télécharger des observations. Plus récemment, des stations de communication tierces utilisant le paradigme de la station au sol en tant que service (GSaaS) ont été développées [1]. Dans ce paradigme, les clients réservent des ressources de communication auprès d'un fournisseur de segments terrestres au lieu de construire leurs propres stations.

Une couche de fédération multi-agents pour coordonner plusieurs missions d'observation de la Terre indépendantes a été proposée [2]. L'objectif de cette fédération est de permettre aux clients qui demandent l'acquisition de vastes zones d'accéder facilement à plusieurs constellations de satellites et de sites de communication afin de composer et de télécharger leurs acquisitions en un temps réduit comparé aux demandes conventionnelles non coordonnées. Le système de communication et de gestion des ressources des satellites (SCRMS) doit gérer les communications en utilisant non seulement les stations terrestres appartenant aux

missions de la fédération, mais aussi le GSaaS.

Le présent travail est consacré au développement et à l'évaluation expérimentale d'algorithmes permettant de résoudre ce problème d'allocation. Sa principale contribution est la fourniture de lignes directrices solides pour la sélection d'algorithmes pour la fonction de réservation de sites de communication.

2 Problème de sélection

Le SCRMS est chargé de fournir des possibilités de communication avec le sol à tous les satellites de toutes les missions fédérées. Il gère la communication entre d'une part N satellites et d'autre part des sites comprenant une ou plusieurs stations au sol, chaque station contrôlant une antenne spécifique. Pour cela il recherche des fenêtres de communication pour le transfert de données en liaison montante et descendante. Les missions fournissent les orbites de leurs satellites et les réseaux de stations au sol publient les coordonnées et les masques de visibilité de leurs stations.

La propagation des orbites et le calcul des événements conduisent pour chaque satellite i à L_i contacts potentiels avec le sol. Chaque contact $l \in \{1, \dots, L_i\}$ est caractérisé par son site au sol, et sa fenêtre temporelle $[\underline{u}_{i,l}, \bar{u}_{i,l}]$, permettant de déduire sa durée $d_{i,l}$. La variable de décision associée à un contact est $x_{i,l} \in \{0, 1\}$, valant 1 si le contact l du satellite i est choisi dans la réservation.

La sélection de contacts permet de remplir en partie les besoins de communications d'un satellite. Chaque satellite i a K_i besoins à remplir, définis par une durée de communication $D_{i,k}$, à remplir sur une fenêtre temporelle $[\underline{t}_{i,k}, \bar{t}_{i,k}]$, avec une liste de sites réservables $S_{i,k}$ et une bande de radiocommunication (bande S pour téléverser les futurs plan d'observation et les opérations de maintenance, et bande X pour le téléchargement des observations). Un contact peut donc être choisi pour participer à la validation d'un besoin k si sa fenêtre temporelle est incluse dans celle du besoin, que le site fait partie des sites réservables et que la bande de radiocommunication est la bonne. Notons $\mathcal{Z}_{i,l}$ l'ensemble des besoins qu'un contact l est en mesure de satisfaire.

Notre problème de décision est donc celui de sélectionner des contacts tel que le plan obtenu puisse vérifier pour chaque satellite i et chacun de ses besoins k

$$\sum_{l \in \{1, \dots, L_i\} / k \in \mathcal{Z}_{i,l}} d_{i,l} x_{i,l} \geq D_{i,k}.$$

Nous pouvons ainsi optimiser deux critères : le coût total

*Ce document résume en français un article accepté à la conférence CPAIOR'25 [3].

de la sélection, en additionnant le coût des contacts retenus (dépendant du modèle de coût du GSaaS, basé sur la durée du contact ou sur un coût fixe par contact), et le conflit et brouillage total induit par la sélection. Deux contacts l et m des satellites i et j , respectivement, peuvent générer une quantité $b_{i,l,j,m}$ de conflit et brouillage si leurs fenêtres temporelles se chevauchent et qu'ils partagent le même site (pas nécessairement la même antenne). Le total est obtenu en faisant la somme du conflit et brouillage généré par toutes les paires de contacts. Nous considérons ici un ordre lexicographique sur les critères, nous minimisons en premier le coût (resp. le conflit et brouillage) et à valeur égale nous minimisons le conflit et brouillage (resp. le coût).

3 Méthodes de résolution

Round-robin. La procédure round-robin permet d'envisager une distribution équitable d'articles entre des agents. Nous associons ici les besoins aux agents et les contacts aux articles. Chaque besoin est considéré à tour de rôle et sélectionne un contact supplémentaire. L'ordre d'examen des besoins dépend de l'évaluation de ces derniers ou de leur difficulté, déclinant deux méthodes. L'évaluation du besoin k du satellite i s'obtient en sélectionnant la meilleure valeur sur les critères calculée en ajoutant un des contacts non sélectionnés qui peuvent contribuer au besoin k . La dureté du besoin est proportionnelle au ratio de sa durée de communication et de la durée totale possible parmi les contacts non sélectionnés.

Procédures gloutonnes. Un schéma d'allocation glouton sélectionne les contacts un par un en faisant le choix qui semble le meilleur sur le moment, sans réviser cette décision par la suite. Nous considérons deux manières de choisir : soit l'on se concentre sur les besoins, soit les contacts. Dans le premier cas, les besoins sont sélectionnés de manière itérative pour être pleinement satisfaits, guidé par leur évaluation ou par leur dureté (similaire au Round-robin). La deuxième méthode conduit à un glouton guidé par la sélection du contact avec la meilleure évaluation.

Recherches locales. Les procédures round-robin et gloutonnes sont des heuristiques constructives : elles produisent des solutions partielles à chaque étape jusqu'à ce qu'elles atteignent l'admissibilité. À l'inverse, les recherches locales travaillent sur des solutions complètes en essayant d'améliorer les critères tout en maintenant l'admissibilité. Ces méthodes nécessitent une solution initiale et la définition du voisinage d'une solution. Pour choisir la solution initiale nous nous servons de deux heuristiques : si le coût est optimisé en premier nous sélectionnons pour chaque satellite i les contacts par ordre croissant de coût jusqu'à ce que ses besoins soient remplis, et sinon nous utilisons la procédure du glouton guidé par l'évaluation du besoin. Avec un voisinage obtenu par la dé-sélection d'un unique $x_{i,l}$ (en interdisant celles qui violent la contrainte sur les besoins), nous définissons deux schémas de recherche : une recherche du meilleur d'abord, qui sélectionne la solution avec la meilleure évaluation parmi toutes les solutions déjà évaluées mais non développées, et une recherche

en profondeur d'abord, qui sélectionne la meilleure solution voisine de la solution courante. Avec un plus grand voisinage obtenu en dé-sélectionnant un nombre donné de contacts nous pouvons également reconstruire l'admissibilité à l'aide d'une recherche à large voisinage (*Large Neighborhood Search*).

Programmation linéaire en nombres entiers. Les contraintes sur les besoins ainsi que l'expression du coût sont déjà linéaires et s'intègrent directement. L'expression du conflit et du brouillage est cependant quadratique, étant donné qu'elle contient le produit $x_{i,l}x_{j,m}$ afin d'ajouter la quantité $b_{i,l,j,m}$ ssi les deux contacts l et m sont sélectionnés. Cette expression est cependant facilement linéarisable sans approximation en introduisant une variable binaire intermédiaire $y_{i,l,j,m}$, remplaçant ce produit, et trois contraintes linéaires pour chaque paire de contacts.

4 Résultats et perspectives

Les performances sont comparées sur un scénario réaliste de 60 jours, avec une fédération de 3 constellations de 2,4 et 4 satellites respectivement, et des besoins qui évoluent tout au long de cette période. Les méthodes doivent planifier chaque jour les contacts des 10 prochains. Les résultats obtenus sont satisfaisants, surtout pour la modélisation linéaire qui respecte la contrainte de temps de calcul imposée (moins de 5 minutes pour obtenir le plan). Certains aspects du problème réel ne sont cependant pas encore pris en compte, comme la possibilité que le GSaaS refuse un contact. Il est aussi possible d'avoir des forfaits de communication avec un nombre de contacts prépayé et un coût supplémentaire pour chaque contact excédent. De plus les besoins considérés sont routiniers, dont seul la charge évolue, mais il existe aussi des besoins urgents, liés à l'observation d'une zone spécifique. Il est alors nécessaire de réserver des contacts avant et après le survol de cette zone pour mettre à jour les plans d'observation et récupérer ces dernières le plus rapidement possible. Ces travaux sont réalisés dans le projet européen DOMINO-E (<https://domino-e.eu/>).

Références

- [1] Elisa Carcaillon and Berylia Bancquart. Market perspectives of ground segment as a service. *71st International Astronautical Congress (IAC)*, 2020.
- [2] Jean-Loup Farges, Filippo Studzinski Perotto, Cédric Pralet, Gauthier Picard, Cyril de Lussy, Jonathan Guerra, Philippe Pavero, and Fabrice Planchou. Going beyond mono-mission earth observation : Using the multi-agent paradigm to federate multiple missions. In *23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS-24)*, 2024.
- [3] Héoïk Willot, Jean-Loup Farges, Gauthier Picard, and Philippe Pavero. Satellite communication resources management in a earth observation federation of constellations. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR-25)*, 2025.

Modèle de programmation par contraintes pour l'équilibrage et l'ordonnancement des lignes d'assemblage avec travailleurs mobiles et stations parallèles

Xavier Pucel¹, Stéphanie Roussel¹

¹ DTIS, ONERA, Université de Toulouse, France

{xavier.pucel,stephanie.roussel}@onera.fr

Résumé

Dans le contexte des lignes d'assemblage en aéronautique, l'augmentation du taux de production et la réduction des coûts d'exploitation sont deux objectifs importants, mais parfois contradictoires. Dans les petites lignes d'assemblage, le partage des ressources de production entre les stations de travail constitue un moyen simple et efficace de réduire les coûts d'exploitation. Ainsi, les travailleurs ne sont pas affectés à une unique station, mais peuvent se déplacer entre elles. De plus, la parallélisation des stations de travail est une approche efficace pour augmenter le taux de production. Toutefois, la combinaison de ces deux stratégies complexifie la planification du travail sur la ligne. Cet article traite du problème de l'affectation des tâches aux stations de travail et de leur ordonnancement dans une ligne d'assemblage où les travailleurs peuvent se déplacer entre les stations, et où ces dernières peuvent être organisées en parallèle. Nous modélisons ce problème à l'aide de la programmation par contraintes. Nous l'évaluons sur des cas d'étude industriels réels provenant de fabricants d'avions, ainsi que sur des jeux de données basés sur des instances la littérature.

Mots-clés

Programmation par contraintes, Ligne d'assemblage, Équilibrage et ordonnancement, Stations de travail parallèles, Travailleurs mobiles

Abstract

In the context of aircraft assembly lines, increasing the production rate and decreasing the operating costs are two important, and sometimes contradictory, objectives. In small assembly lines, sharing production resources across workstations is a simple and efficient way to reduce operating costs. Therefore, workers are not assigned to a unique workstation but can walk between them. On the other side, paralleling workstations is an efficient way to increase the production rate. However, the combination of both strategies create complex conditions for tasks to access the production resources. This paper addresses the problem of allocating tasks to workstations and scheduling them in an assembly line where workers can walk across workstations,

and where workstations can be organized in parallel. We model this problem with Constraint Programming. We evaluate it on real world industrial use cases coming from aircraft manufacturers, as well as synthetic use cases adapted from the literature.

Keywords

Constraint Programming, Assembly Line, Balancing and Scheduling, Parallel Workstations, Walking Workers

1 Description du problème

La production d'avions est souvent réalisée sur des chaînes d'assemblage. Pour augmenter le taux de production, il est possible d'ajouter des stations de travail ou d'organiser des stations en parallèle. Cependant, ces approches présentent des inconvénients tels que l'occupation d'un espace supplémentaire en usine et des coûts élevés.

Cet article est un résumé du travail [2], qui peut être considéré comme une extension du *Resource Constrained Project Scheduling Problem* (RCPSPP) avec des stations de travail et des ressources cumulatives partagées entre stations. Nous nous intéressons au problème de l'optimisation du temps de cycle (i.e. le temps passé sur chaque station de travail) sur une chaîne d'assemblage pulsée, qui vient d'un cas d'étude réalisé avec un avionneur. Une ligne pulsée est découpée en plusieurs stations de travail. Ainsi, après chaque temps de cycle, l'avion passe de sa station courante à la suivante sur la chaîne. Le problème considéré présente les caractéristiques suivantes.

- On souhaite minimiser la durée d'assemblage totale.
- Le nombre et la disposition des postes de travail sont donnés en entrée. Aussi, le seul moyen de diminuer la durée d'assemblage est de réduire le temps de cycle.
- La station est découpée en zones dans lesquelles les tâches ne peuvent pas se chevaucher temporellement.
- Les techniciens peuvent se déplacer entre les stations de travail, sans durée de transition.
- Chaque technicien a une compétence spécifique et les tâches nécessitent un nombre précis de techniciens par compétence. Les techniciens réalisent au maximum une tâche à chaque instant.
- Les tâches ne peuvent pas être interrompues. Elles sont

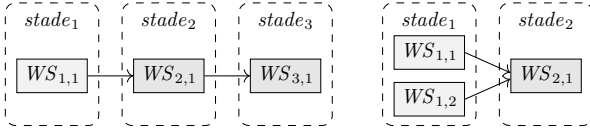


FIGURE 1 – Illustration de deux configurations d'usine

de plus assignées à une unique station de travail.

La ligne d'assemblage est divisée en *stades*, qui peuvent chacun contenir une ou plusieurs stations de travail en parallèle. La figure 1 illustre deux configurations de ligne. Dans la première configuration (à gauche sur la figure), chaque stade contient une unique station de travail et l'avion passe exactement la même durée dans chaque station de travail. Dans la deuxième configuration (à droite), le premier stade est composé de deux stations de travail en parallèle permettant à l'avion de passer deux temps de cycle sur chacune de ces stations. Dans cette configuration, la station de travail du deuxième stade, $WS_{2,1}$, est alimentée alternativement par des avions venant $WS_{1,1}$ et $WS_{1,2}$.

Plusieurs travaux de la littérature sont en lien avec ce problème. [1] décrit une approche d'*Assembly Line Balancing Problem* (ALBP) avec stations parallèles, affecte les tâches aux stations de travail, mais ne les ordonne pas. [3] présente une approche de *Multi-Manned ALBP* avec travailleurs mobiles, affecte les tâches aux stations et les ordonne, mais ne supporte pas les stations parallèles. L'originalité de notre approche est la considération conjointe de techniciens mobiles et de stations de travail parallèle. En effet, dans cette configuration, une tâche peut se retrouver en compétition avec elle-même pour l'accès aux ressources mobiles. La figure 2 illustre ce cas de figure : 8 tâches doivent être réalisées sur la ligne avec la configuration de droite de la figure 1, et une ressource R_1 représentant trois techniciens mobiles requis pour les tâches t_1 , t_2 et t_6 . Dans cet exemple, la tâche t_2 est à cheval sur le temps de cycle et deux techniciens peuvent être mobilisés simultanément pour la réaliser (sur des avions qui se suivent sur la ligne).

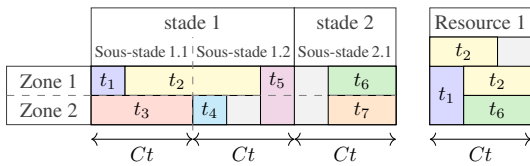


FIGURE 2 – Exemple de solution pour un problème à 8 tâches dans la configuration parallèle

2 Contributions et expérimentations

Définition formelle du problème. Nous avons tout d'abord défini formellement une extension du problème de ligne d'assemblage avec ordonnancement des tâches, mobilité des techniciens et stations parallèles.

Modèle PPC. Un modèle en Programmation par Contraintes a été proposé. Ce modèle s'appuie sur de multiples variables de décision de type intervalle organisées

hiérarchiquement, permettant de représenter les affectations de tâches (choix du stade, choix de la station) et leur consommation des ressources. Plus précisément, chaque tâche est associée à une variable intervalle obligatoire correspondant à sa réalisation et plusieurs variables intervalles optionnelles pour représenter les assignations dans chaque stade possible. Si un stade est composé de plusieurs sous-stade, chaque intervalle optionnel est lui-même décomposé en plusieurs intervalles pour représenter la réalisation de la tâche sur chaque sous-stade. Par exemple, la tâche t_2 de la figure 2 a une première partie sur le sous-stade 1.1 et une seconde partie sur le sous-stade 1.2. Finalement, ces intervalles optionnels peuvent être décalés temporellement et être utilisés pour exprimer la consommation des ressources partagées. Sur l'exemple, le sous-intervalle de t_2 sur le sous-stade 1.1 peut-être utilisé tel que pour la consommation de la ressource 1. Cependant, le sous-intervalle de t_2 sur le sous-stade 1.2 doit être décalé d'un temps de cycle. Les sous-intervalles résultant font ensuite l'objet de contraintes de type *Cumulative* pour chaque ressource. La description complète du modèle est disponible dans [2].

Expérimentations. Nous avons réalisé des expérimentations sur plusieurs jeux de données :

- des instances réelles venant d'un avionneur avec une ligne composée de stations parallèles ;
- des instances adaptées d'un jeu de données venant d'un avionneur mais pour un autre problème ;
- des instances de la littérature (PSPLib) adaptées pour correspondre au problème traité.

Le modèle PPC a été implémenté et testé avec l'outil ILOG Cplex Optimizer 20.1. Le solveur trouve la solution optimale sur des petits jeux de données. Sur les plus grosses instances, le solveur trouve une solution rapidement mais ne peut pas prouver son optimalité.

Analyse et perspectives. Globalement, la parallélisation des tâches permet de diminuer le temps de cycle. Néanmoins, un aspect non pris en compte ici est la surface au sol nécessaire pour la duplication des stations. Il faudrait donc étendre ce travail en considérant un (des) critère(s) supplémentaire(s) pour contrebalancer le gain à paralléliser.

Références

- [1] Eduardo Álvarez-Miranda, Sebastián Chace, and Jordi Pereira. Assembly line balancing with parallel workstations. *International Journal of Production Research*, 59(21) :6486–6506, 2021.
- [2] Xavier Pucel and Stéphanie Roussel. Constraint programming model for assembly line balancing and scheduling with walking workers and parallel stations. In *CP 2024*, volume 307, pages 23 :1–23 :21, 2024.
- [3] Murat Şahin and Talip Kellegöz. Balancing multi-manned assembly lines with walking workers : problem definition, mathematical formulation, and an electromagnetic field optimisation algorithm. *International Journal of Production Research*, 57(20) :6487–6505, 2019.

Remplissage de camions citerne, routage et gestion d'inventaire d'hydrocarbures avec OptalCP

Diego Olivier Fernandez Pons, Petr Vilim
OptalCP

Résumé

Nous résolvons un problème de remplissage de camions citerne, transport et gestion d'inventaire d'hydrocarbures. Le problème combine ordonnancement, affectation de tâches à des shifts, emplois du temps de personnel, gestion de niveau d'inventaires et routage. Il illustre les variantes complexes de flexible-jobshop que l'on rencontre dans la pratique. Le problème étant trop complexe pour être résolu directement, nous procédons à des simplifications qui sont discutées et évaluées dans ce travail.

Mots-clés

Ordonnancement, routage, inventaire, programmation par contraintes, optalcp

Abstract

We solve a problem from the oil industry that combines scheduling, transportation with shifts, inventory management and routing. This problem is an example of the complex variants of the flexible jobshop problem that appear in industrial applications. The original problem cannot be solved in all its complexity with existing scheduling engines hence we had to evaluate various simplifications of the problem. The paper details the modeling process.

Keywords

Scheduling, routing, inventory management, constraint programming, optalcp

1 Présentation du problème

1.1 Production et transport d'hydrocarbures

L'extraction, raffinage et distribution d'hydrocarbures sont usuellement séparés en opérations *upstream*, tout ce qui est antérieur à la raffinerie, donc l'extraction de pétrole et son transport par oléoduc, et en opérations *downstream*, qui relèvent de la distribution de produits finis par oléoduc, bateau, train ou camion citerne.

Le transport downstream ressemble à un arbre où les niveaux successifs transportent des volumes de plus en plus faibles à des points de distribution de plus en plus nombreux : un oléoduc pourra par exemple desservir 2 ports depuis la raffinerie, des bateaux assureront l'acheminement vers une dizaine d'autres ports, où le pétrole sera transporté par camion vers une centaine de points de distribution (des villes) d'où il sera acheminé aux points de vente finaux (stations essence).

Tous les problèmes de transport d'hydrocarbures comportent une dimension de gestion d'inventaire : les points en aval consomment des hydrocarbures en permanence (et de façon variable en fonction de critères divers comme la saisonnalité), pendant que les points en amont produisent des hydrocarbures. Et la capacité de stockage est finie en tout point du réseau.

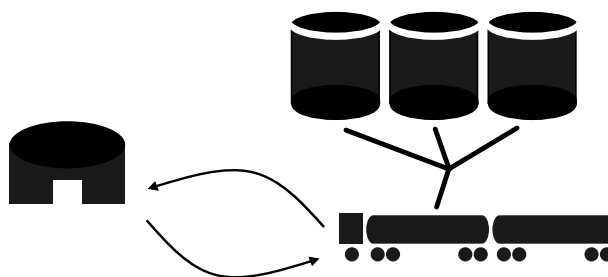


FIGURE 1 – Camion citerne à double cuve dans une ligne de remplissage ayant accès à 3 réservoirs, puis transport et décharge chez un client.

1.2 Distribution par camions citerne

Les centres de distribution planifient sur une fenêtre glissante de 14 jours de la façon suivante :

- Les camions citerne font la queue dans des lignes de distribution afin de remplir leurs cuves, ensuite ils se dirigent vers un client à livrer, déchargent après possiblement une attente, puis reviennent au centre de distribution
- Les lignes de remplissage ont des horaires de fonctionnement et peuvent distribuer plusieurs types d'hydrocarbures, avec un temps supplémentaire si changement d'hydrocarbure
- Certains clients ont des fenêtres de livraison et des installations de décharge particulières (nombre de tuyaux, volume maximal)
- Les conducteurs ont des contraintes horaires (shift de 7h + heures supplémentaires, 12h de repos minimum, 1 shift par jour maximum, 2 x 2 jours de repos consécutifs)

Dans le problème de distribution secondaire (du port vers les villes) qui nous concerne les camions ont deux (2) cuves (figure 1)

2 La solution existante

De nombreuses contraintes présentes dans la solution existante ne sont pas motivées par des limitations techniques mais par le besoin de simplifier le problème pour rendre sa résolution plus rapide. Nous appelons *approximation* une contrainte qui modifie ainsi le problème.

2.1 Approximations utilisées

Les approximations suivantes étaient déjà présentes dans la solution que le client désire remplacer

2.1.1 Simplification de la gestion d'inventaire

La motivation de ces simplifications est de réduire la taille des problèmes à résoudre en le coupant en journées.

— Séparation du problème en journées

L'horizon de 14 jours est divisé en 13 problèmes de 24h et les livraisons sont réparties uniformément. Les livraisons qui n'ont pu être réalisées au jour j sont reportées au jour $j + 1$.

2.1.2 Simplification du remplissage des camions

La motivation de ces simplifications est de rendre le problème d'ordonnancement plus simple.

— Les réservoirs ont un volume illimité

Les lignes de remplissage sont connectées à des réservoirs qui contiennent chacun un type d'hydrocarbure (figure 1). Si la capacité de ces réservoirs est considérée illimitée, le problème se réduit à des lignes de remplissage qui ont une liste de types d'hydrocarbures qu'elles peuvent utiliser pour remplir les cuves des camions citerne.

— Les temps de remplissage des cuves est constant

Bien que très légèrement différent pour chaque type d'hydrocarbure en raison de la différence de viscosité, on peut le considérer constant.

— Le temps de changement de camion dans une ligne de remplissage est constant

Il peut être inclus dans la durée de remplissage.

— Le temps de changement d'hydrocarbure dans une ligne de remplissage est constant

On fixe ce temps à 10 minutes indépendamment du type d'hydrocarbure.

— Le temps de changement par un camion d'une ligne à une autre est constant

On néglige la distance entre les lignes de remplissage.

2.1.3 Simplification du transport

Ces simplifications évitent les variantes complexes de problèmes de transport (selection de clients, multi-stop, gestion des quais, emplois du temps).

— Clients transportant leurs hydrocarbures

Les centres de distribution n'ayant pas la capacité de livrer à tous les clients, un accord est en place avec certains clients pour qu'ils cherchent leur hydrocarbures au centre de distribution avec leur propres camions (dont la taille et capacité sont hétérogènes).

— Les deux cuves d'un camion citerne sont destinées au même client

Cette limitation permet de faire du routage en étoile au lieu de routage "multi-stop".

— Les clients n'ont qu'un tuyau de décharge

Si un client a une capacité supérieure, il est dupliqué et les livraisons sont réparties entre les deux clients virtuels. Cela permet d'avoir un problème disjonctif à la décharge plutôt que cumulatif.

— Pas de temps de préparation entre décharges

Ce temps est négligé à la décharge.

2.1.4 Simplification des contraintes légales de travail

Un centre de distribution a un nombre fixe de camions et un nombre modifiable de conducteurs. En toute généralité il faudrait attribuer à la fois des livraisons et un camion à chaque shift de conducteur puis ajouter les contraintes légales de travail et enfin des temps morts pour la maintenance des camions. Mais comme le problème est divisé en journées, les camions et les conducteurs ont été fusionnés en un "shift par camion par jour".

— Un shift par camion par jour

Résultat de la fusion du conducteur et du camion.

— Les shifts sont limités à 12h

Un shift fait 7h + heures supplémentaires à distribuer avec précaution sur l'ensemble des conducteurs. La limite de 12h est une première approche à réviser selon les résultats.

— Pas de temps de repos

Les conducteurs ont le droit à plusieurs temps de repos (break) pendant leur shift. Ces temps de repos ont été ignorés car la solution actuelle laissait des "trous" suffisants pour les ajouter en post-traitement.

— Pas de durée minimale entre les shifts

Les conducteurs ont droit à 12h de repos entre les shifts. Cette contrainte est ignorée dans le problème journalisé.

— Pas de journée repos

Les conducteurs de camion ont le droit à des journées de repos alternant avec les journées de travail. Ces contraintes sont ignorées dans le problème journalisé.

2.2 Solution par fenêtre glissante

La solution complète opère de la façon suivante

1. L'horizon de 14 jours est découpé en 13 problèmes journaliers par pré-allocation uniforme des livraisons de chaque client aux journées.
2. Les problèmes journaliers sont résolus en séquence en reportant au jour $j + 1$ les livraisons qui n'ont pas pu être satisfaites pendant le jour j .
3. Les 13 journées sont assemblées et les shifts de 12h qui amalgament camion et conducteur sont attribués à des conducteurs en respectant les contraintes légales. Tout ce qui provoque des violations lors de l'assemblage est envoyé en 14e journée. Enfin les clients devant chercher leurs hydrocarbures au centre de distribution sont enfin ajoutés dans tous les lignes de chargement.

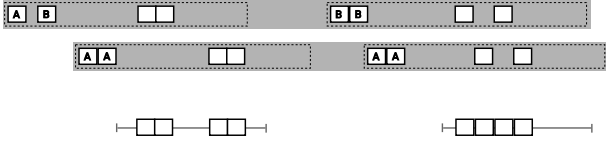


FIGURE 2 – Deux camions en gris (Truck) faisant chacun 2 voyages en pointillés (Trip). Chaque voyage comporte le remplissage des 2 cuves du camion citerne avec les hydrocarbures de type A ou B (Load), et deux décharges chez le client (Unload). En dessous les mêmes décharges à l'intérieur des fenêtres temporelles

2.3 Formalisation

Nous formalisons le problème journalier tel que résolu par la solution existante au moyen du langage d'ordonnancement de OptalCP [1] (similaire à celui de IBM CP Optimizer [2]).

Cette formalisation est issue de discussions et d'observations, nous ne pouvons garantir que l'outil en place adhère complètement à cette spécification.

2.3.1 Modèle journalier de référence

Dans le problème il y a deux chargements et deux déchargements par voyage (figure 2). Afin de ne pas complexifier davantage la formulation, et parce que c'est la première simplification faite par la suite, notre modèle de référence ne fait apparaître qu'un seul chargement et déchargement.

Constantes

- Les livraisons pour le client c sont indexées par $k \in [1 \dots N^c]$
- Le nombre de livraisons du produit p pour chaque client c est N_p^c avec $\sum_p N_p^c = N^c$
- M_p^c est le nombre de livraisons pour le client c de l'hydrocarbure p qui n'ont pas pu être livrées la veille est sont obligatoires
- Nous appelons LP les paires (ligne, produit) compatibles, L_p les lignes compatibles avec le produit p et P_l les produits disponibles à la ligne l
- T_c est le temps pour conduire du centre de distribution au client c
- La matrice $setup$ est le temps de changement d'hydrocarbure dans les lignes, avec 10 minutes si les hydrocarbures sont de type différent, 0 sinon

Variables

- $Load^{c,k}$ la charge optionnelle de la k -ième livraison pour le client c
- $Load_l^{c,k}$ la charge optionnelle sur la ligne l de la k -ième livraison pour le client c
- $Load_p^{c,k}$ la charge optionnelle en produit p de la k -ième livraison pour le client c
- $Unload^{c,k}$ la décharge optionnelle de la k -ième livraison pour le client c
- $Trip^{c,k}$ le k -ième voyage optionnel pour le client c
- $Trip_t^{c,k}$ le k -ième voyage optionnel pour le client c opéré sur le camion t
- $Truck_t$ la tâche optionnelle qui représente l'ensemble des voyages du camion t

Équations

$$\max \sum_{c,k} \text{presence}(\text{Trip}^{c,k})$$

$$\forall c k \quad \text{alternative}(\text{Load}^{c,k}, \{\text{Load}_l^{c,k} \mid \forall l\}) \quad (1)$$

$$\forall c k \quad \text{alternative}(\text{Load}^{c,k}, \{\text{Load}_p^{c,k} \mid \forall p\}) \quad (2)$$

$$\forall c k \quad \text{presence}(\text{Load}^{c,k}) = \sum_{l,p \in LP} \text{presence}(\text{Load}_l^{c,k}) \times \text{presence}(\text{Load}_p^{c,k}) \quad (3)$$

$$\forall c p \quad M_p^c \leq \sum_k \text{presence}(\text{Load}_p^{c,k}) \leq N_p^c \quad (4)$$

$$\forall l \quad \text{noOverlap}(\{\text{Load}_l^{c,k} \mid \forall c k\}, \text{setup}) \quad (5)$$

$$\forall c k l \quad \text{Load}_l^{c,k} \in \text{Operating hours}_l \quad (6)$$

$$\forall c k \quad \text{start}(\text{Trip}^{c,k}) = \text{start}(\text{Load}^{c,k}) \quad (7)$$

$$\forall c k \quad \text{end}(\text{Load}^{c,k}) + T_c \leq \text{start}(\text{Unload}^{c,k}) \quad (8)$$

$$\forall c k \quad \text{end}(\text{Unload}^{c,k}) + T_c = \text{end}(\text{Trip}^{c,k}) \quad (9)$$

$$\forall c k \quad \text{presence}(\text{Trip}^{c,k}) = \text{presence}(\text{Load}^{c,k}) \quad (10)$$

$$\forall c k \quad \text{presence}(\text{Trip}^{c,k}) = \text{presence}(\text{Unload}^{c,k}) \quad (11)$$

$$\forall c \quad \text{noOverlap}(\{\text{Unload}^{c,k} \mid \forall k\}) \quad (12)$$

$$\forall c k \quad \text{Unload}^{c,k} \in \bigcup \text{Time Window}_c \quad (13)$$

$$\forall c k \quad \text{alternative}(\text{Trip}^{c,k}, \{\text{Trip}_t^{c,k} \mid \forall t\}) \quad (14)$$

$$\forall t \quad \text{noOverlap}(\{\text{Trip}_t^{c,k} \mid \forall c k\}) \quad (15)$$

$$\forall t \quad \text{Truck}_t = \text{span}(\{\text{Trip}_t^{c,k} \mid \forall c k\}) \quad (16)$$

$$\forall t \quad \text{size}(\text{Truck}_t) \leq 12 \text{ hours} \quad (17)$$

Le problème peut être vu comme la fusion de plusieurs problèmes classiques

Flexible jobshop (1-6)

(1) et (2) exige que pour chaque tâche Load soient choisis une ligne et un produit qui sont obligés d'être compatibles par (3). Le nombre de livraisons d'un produit à un client ne doit pas être supérieur au nombre qu'il a commandé ni inférieur au nombre de livraisons abandonnées la veille (4). (5) impose que chargements qui ont lieu sur une ligne ne se chevauchent pas, et (6) qu'ils aient lieu aux heures d'ouverture de la ligne.

Span complexe (7-11)

(10) et (11) imposent que si une livraison a lieu, elle comporte un chargement et un déchargement (et inversement). (7) aligne le chargement et le début de la livraison, (8) qu'il y ait assez de temps entre le chargement et le déchargement pour se déplacer chez le client, (9) que le camion retourne au centre à peine son déchargement fini.

Livraison avec fenêtres temporelles (12-13)

(12) impose aux décharges de ne pas se chevaucher, (13) de s'effectuer pendant les fenêtres d'ouverture.

Bin packing temporel (14-17)

(14) exige que pour chaque tâche Trip soit choisi un camion, (15) que les voyages d'un camion ne se chevauchent pas, (16) que le shift d'un camion couvre exactement ses voyages et (17) que le shift d'un camion dure au maximum 12 heures.

2.3.2 Fonction objectif

Livraisons de la veille

L'objectif pousse le moteur à faire le maximum de livraisons possibles durant la journée tandis que la borne inférieure de la contrainte (4) garantit que les livraisons ignorées de la journée précédente seront nécessairement livrées dans la journée courante. Comme les livraisons d'un même produit sont interchangeables, cette contrainte s'exprime sur la somme et non sur les livraisons individuelles.

Dans les jeux de données fournis on considère $M_p^c = 0$, autrement dit la borne inférieure de la contrainte (4) est ignorée pour que les 13 journées puissent être testées indépendamment.

Équilibrage des heures supplémentaires

La durée de 12h pour les shifts est la durée maximale légale, en réalité les shifts font 7h + heures supplémentaires qu'il faudrait distribuer de façon équilibrée au moyen d'un objectif **leximin**. Comme le modèle journalier ne représente pas les chauffeurs, et que les objectifs de maximisation du nombre de livraisons et limitation des heures supplémentaires sont directement en conflit, cet objectif est ignoré.

2.3.3 Modèle d'assemblage des journées

Les solutions des 13 modèles journaliers forment un ensemble I d'intervalles indexés par les journées j et les camions t . Afin d'y associer un conducteur d , on introduit les variables booléennes $x_{jt}^d \in \{0, 1\}$ et le modèle linéaire

$$\max \sum_d \sum_j \sum_t x_{jt}^d$$

$$\forall d \quad \forall j \quad \sum_t x_{jt}^d \leq 1 \quad (18)$$

$$\forall d \quad \sum_{jt} x_{jt}^d \leq 10 \quad (19)$$

$$\forall d \quad \forall j_1 \quad j_2 \quad t_1 \quad t_2 \quad \text{distance}(I_{j_1 t_1}, I_{j_2 t_2}) \leq 12 \text{ hours} \\ \Rightarrow x_{j_1 t_1}^d + x_{j_2 t_2}^d \leq 1 \quad (20)$$

La contrainte (18) impose à tout conducteur au plus un shift par jour, tandis que (19) indique que le conducteur a 4 jours de repos. La contrainte (20) empêche qu'un conducteur travaille des shifts consécutifs séparés de moins de 12 heures. Les shifts qui n'auront pas pu être attribués sont reversés dans la 14e journée.

2.4 Limitations du modèle journalier

La limitation la plus sévère du modèle journalier est que le concept de conducteur ne peut pas être représenté directement, si bien que les contraintes légales liées aux conducteurs ne sont pas considérées. Quand l'algorithme d'assemblage identifie une impossibilité due à une contrainte liée aux conducteurs, il est trop tard pour remettre en cause la solution du modèle journalier.

Le choix de cette décomposition malgré ces limitations est motivé par la nécessité de réduire autant que possible la complexité du problème afin qu'il soit soluble en un temps raisonnable.

2.5 Plan de travail

Nos objectifs dans ce travail sont

1. Réduire le temps de résolution du problème journalier au moyen d'approximations. Ces approximations modifient le problème étudié (autrement dit éliminent des solutions auparavant admissibles).
2. Réduire le temps de résolution par une étude de différents modèles. Ces modèles améliorent la vitesse de résolution sans altérer la nature du problème étudié.
3. Essayer de transformer le modèle journalier ainsi amélioré en un modèle multi-jours afin d'y ajouter les considérations liées aux conducteurs.

3 Nouvelles approximations pour le problème journalier

Nous allons considérer les approximations suivantes

- La réduction de 2 cuves à une cuve double
- L'élimination des temps d'attente avant la décharge
- La spécialisation des lignes de remplissage

3.1 Réduction de deux cuves à une

Appelons une *configuration* le couple d'hydrocarbures qui est chargé dans les cuves d'un camion citerne, par exemple AA, AB, BA, BB, etc.

- Si l'on impose que les deux cuves transportent le même produit alors le nombre de configurations requises de chaque type possible est connu à l'avance. Par exemple la seule façon de livrer 6 ordres de produit A et 4 ordres de produit B est $6A + 4B = 3AA + 2BB$.
- Au contraire si les contenu des 2 cuves est indépendant, le nombre de configurations ne peut être déterminé à l'avance. Par exemple $6A + 4B$ peut être aussi obtenu par $2AA + AB + BA + BB$ ou encore $4AB + AA$.

Remplacer les doubles cuves par une cuve unique de double volume, permet de remplacer la suite d'intervalles $\text{Load}^{c,k}$ avec $k \in [1 \dots \sum_p N_p^c]$ par autant de suites d'intervalles que de produits $\text{Load}^{c,p,k}$ avec $k \in [1 \dots N_p^c]$ (figure 3).

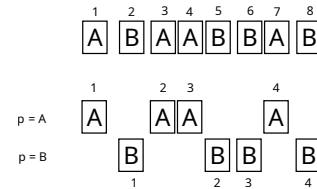


FIGURE 3 – Avoir une seule cuve permet de créer une séquence par type de produit

3.1.1 Modèles sans double alternative

La réduction de deux cuves à une permet l'utilisation de modèles sans la double alternative. Avec les variables $\text{Load}^{c,k}$ $k \in [1 \dots \sum_p N_p^c]$ il faut choisir la ligne et le produit (double alternative) puis garantir à la fois leur compatibilité et le nombre total de produits livrés. Dans le modèle

de référence les équations correspondantes sont :

$$\forall c k \quad \text{alternative}(\text{Load}^{c,k}, \{\text{Load}_l^{c,k} \mid \forall l\}) \quad (1)$$

$$\forall c k \quad \text{alternative}(\text{Load}^{c,k}, \{\text{Load}_p^{c,k} \mid \forall p\}) \quad (2)$$

$$\forall c k \quad \text{presence}(\text{Load}^{c,k}) == \sum_{l,p \in LP} \text{presence}(\text{Load}_l^{c,k}) \times \text{presence}(\text{Load}_p^{c,k}) \quad (3)$$

$$\forall c p \quad \sum_k \text{presence}(\text{Load}_p^{c,k}) \leq N_p^c \quad (4)$$

Avec les variables $\text{Load}^{c,p,k}$ $k \in [1 \dots N_p^c]$ il faut juste choisir la ligne. Le choix du produit disparaît, la compatibilité est pré-calculée via L_p et le nombre de livraisons du produit p est par construction $\leq N_p^c$

$$\forall c p k \quad \text{alternative}(\text{Load}^{c,p,k}, \{\text{Load}_l^{c,p,k} \mid l \in L_p\})$$

La seconde approche n'est pas nécessairement plus rapide car on perd la possibilité de remplacer de la contrainte `noOverlap` par des précédences après rupture de symétrie. L'étude de ces modèles devra déterminer les gains.

3.2 Élimination du temps d'attente

3.2.1 Durée minimale d'un voyage

Un voyage est défini à partir de ses composants. Les équations correspondantes dans le modèle de référence sont :

$$\forall c k \quad \text{start}(\text{Trip}^{c,k}) = \text{start}(\text{Load}^{c,k}) \quad (7)$$

$$\forall c k \quad \text{end}(\text{Load}^{c,k}) + T_c \leq \text{start}(\text{Unload}^{c,k}) \quad (8)$$

$$\forall c k \quad \text{end}(\text{Unload}^{c,k}) + T_c = \text{end}(\text{Trip}^{c,k}) \quad (9)$$

L'équation (8) est une inégalité car le camion peut arriver au point de livraison et devoir attendre pour la décharge. Inversement l'équation (9) est une égalité car il est inutile d'attendre après la dernière livraison du camion avant de revenir au centre de distribution.

Ces équations impliquent que les voyages ont une durée minimale

$$\text{size}(\text{Trip}^{c,k}) \geq 2T_c + \text{size}(\text{Load}^c) + \text{size}(\text{Unload}^c)$$

3.2.2 Durée maximale d'un voyage

Le produit étant déchargé n'étant pas pertinent pour le problème de décharge, il est inutile qu'un camion arrivé avant chez le client décharge après un camion arrivé après (s'il y avait des temps de changement de produit à la décharge ce ne serait plus le cas). Cette propriété se traduit par :

$$\begin{aligned} \forall c k \quad \text{start}(\text{Trip}^{c,k}) \leq \text{start}(\text{Trip}^{c,k+1}) \Rightarrow \\ \text{start}(\text{Unload}^{c,k}) \leq \text{start}(\text{Unload}^{c,k+1}) \end{aligned}$$

Parce que les décharges sont disjointes, on peut remplacer `start` par `end` dans le second membre de l'implication

$$\begin{aligned} \forall c k \quad \text{start}(\text{Trip}^{c,k}) \leq \text{start}(\text{Trip}^{c,k+1}) \Rightarrow \\ \text{end}(\text{Unload}^{c,k}) \leq \text{start}(\text{Unload}^{c,k+1}) \end{aligned}$$

Ces équations signifient que l'on peut imposer la conservation de l'ordre des départs et des décharges. En absence de fenêtres temporelles, l'attente d'un camion ne peut être que causée par le camion précédent (qui lui même a pu attendre en raison de celui d'avant), donc l'attente du k -ième camion est bornée par $(k-1) \times \text{size}(\text{Unload}^c)$ avec $(k \geq 1)$.

3.2.3 Fixer la durée des voyages

Quand on impose un ordre unique aux charges et décharges, en absence de fenêtres temporelles, les attentes à la décharge sont dues aux départs rapprochés des camions se dirigeant vers un même client (figure 4). Dès lors, imposer que les livraisons se fassent sans attente est équivalent à imposer une distance minimale de `size(Unload)` entre les départs pour un même client. Cela revient à constater qu'une contrainte `noOverlap` ($[x_k \dots x_k + C]$) est équivalente à une contrainte `allMinDistance`(x_k, C).

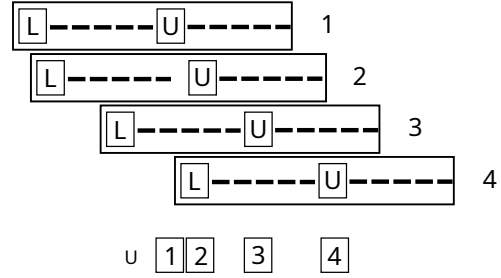


FIGURE 4 – Quatre camions qui livrent un même client, les pointillés désignent le temps de parcours, L les chargements et U les déchargements. On constate que U_2 doit attendre que U_1 soit fini.

3.2.4 Voyages de durée fixe et fenêtres temporelles

La présence de fenêtre temporelles peut avoir pour conséquence que l'approximation utilisant des voyages sans attente rende certaines livraisons impossibles. Les tests devront déterminer si c'est le cas.

3.2.5 Simplification de la structure des voyages

L'absence d'attente fixe la structure des voyages et permet de la simplifier : les seules contraintes agissant sur les décharges sont la contrainte `noOverlap` empêchant les décharges de se chevaucher, et les fenêtres temporelles.

$$\forall c \quad \text{noOverlap}(\{\text{Unload}^{c,k} \mid \forall k\}) \quad (12)$$

$$\forall c k \quad \text{Unload}^{c,k} \in \bigcup \text{Time Window}_c \quad (13)$$

Comme la distance entre `Load` et `Unload` est fixe en l'absence d'attente, les variables de déchargement `Unload` peuvent être éliminées et les contraintes posées sur les variables `Load`

$$\forall c \quad \text{noOverlap}(\{\text{Load}^{c,k} \mid \forall k\}) \quad (21)$$

$$\forall c k \quad \text{Load}^{c,k} \in \bigcup (\text{Time Window}_c - T_c) \quad (22)$$

3.2.6 Bin packing à éléments de taille fixe

Les équations (14 - 17) définissent un problème de bin-packing temporel où les éléments $\text{Trip}^{c,k}$ de taille variable doivent être mis dans les boîtes Truck_t .

$$\forall c k \quad \text{alternative}(\text{Trip}^{c,k}, \{\text{Trip}_t^{c,k} \mid \forall t\}) \quad (14)$$

$$\forall t \quad \text{noOverlap}(\{\text{Trip}_t^{c,k} \mid \forall c k\}) \quad (15)$$

$$\forall t \quad \text{Truck}_t = \text{span}(\{\text{Trip}_t^{c,k} \mid \forall c k\}) \quad (16)$$

$$\forall t \quad \text{size}(\text{Truck}_t) \leq 12 \text{ hours} \quad (17)$$

La taille variable des éléments empêche l'ajout d'une contrainte redondante de packing ([4] [3]). Avec l'approximation $\text{size}(\text{Trip}^{c,k}) = L^c$ les contraintes de packing peuvent être utilisées.

Les contraintes de packing ignorant la dimension temporelle du problème, leur efficacité dépendra de la propension des contraintes temporelles (précédences, etc) à créer des trous (figure 5)



FIGURE 5 – La contrainte de packing n'identifie pas les trous - les espaces dans lesquels aucun autre item ne peut aller, par exemple une précédence entre b et c crée un trou entre a et b .

3.3 Spécialisation des lignes de chargement

La figure (6) montre les produits disponibles sur chaque ligne de chargement. On constate que seules 7 lignes sont multi-produit (0 à 5 et 20).

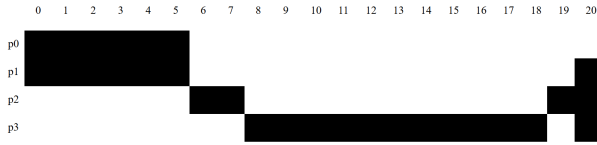


FIGURE 6 – Table de compatibilité entre lignes de remplissage et types d'hydrocarbures

Les avantages des lignes spécialisées sont l'absence de temps de changement de produit et la simplicité de gestion pratique (figure 7)

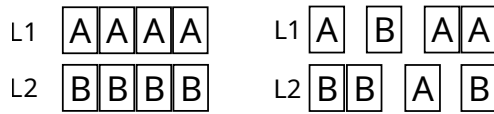


FIGURE 7 – Toutes choses égales par ailleurs, l'opérateur préfère une solution avec moins de changements (setups).

Plusieurs options de complexité croissante sont possibles

1. Spécialiser statiquement les lignes (pre-traitement) en proportion des données.
2. Modifier le modèle pour sélectionner pour chaque ligne un unique type de produit.
3. Modifier le modèle pour minimiser le nombre de changement de produit sur l'ensemble des lignes.

Nous nous limitons ici à la première approche, les deux autres options feront l'objet de travaux ultérieurs.

3.4 Modèle après approximations

Le modèle après approximations, dans la variante double index client c et produit p , s'écrit

Constantes

- Les livraisons du produit p pour le client c sont indexées par $k \in [1 \dots N_p^c]$
- M_p^c est le nombre de livraisons pour le client c de l'hydrocarbure p qui n'ont pas pu être livrées la veille est sont obligatoires
- L_p est l'ensemble des lignes compatibles avec le produit p , et les lignes ne peuvent utiliser qu'un seul type de produit ($p \neq q \Rightarrow L_p \cap L_q = \emptyset$)
- T_c est le temps requis pour se déplacer du centre de distribution au client c

Variables

- $\text{Load}^{c,p,k}$ la charge optionnelle de la k -ième livraison du produit p pour le client c
- $\text{Load}_l^{c,p,k}$ la charge optionnelle sur la ligne l de la k -ième livraison du produit p pour le client c
- $\text{Trip}^{c,p,k}$ la k -ième livraison optionnelle du produit p pour le client c
- $\text{Trip}_t^{c,p,k}$ la k -ième livraison optionnelle du produit p pour le client c opéré sur le camion t
- Truck_t la tâche optionnelle qui représente l'ensemble des voyages du camion t

Équations

$$\max \sum_{c,p,k} \text{presence}(\text{Trip}^{c,p,k})$$

$$\forall c p k \quad \text{alternative}(\text{Load}^{c,p,k}, \{\text{Load}_l^{c,p,k} \mid l \in L_p\}) \quad (23)$$

$$\forall l \quad \text{noOverlap}(\{\text{Load}_l^{c,p,k} \mid \forall c p k\}) \quad (24)$$

$$\forall c p k l \quad \text{Load}_l^{c,p,k} \in \text{Operating hours}_l \quad (25)$$

$$\forall c p \quad M_p^c \leq \sum_k \text{presence}(\text{Load}^{c,p,k}) \quad (26)$$

$$\forall c k \quad \text{start}(\text{Trip}^{c,p,k}) = \text{start}(\text{Load}^{c,p,k}) \quad (27)$$

$$\forall c k \quad \text{presence}(\text{Trip}^{c,p,k}) = \text{presence}(\text{Load}^{c,p,k}) \quad (28)$$

$$\forall c \quad \text{noOverlap}(\{\text{Load}^{c,p,k} \mid \forall p k\}) \quad (29)$$

$$\forall c p k \quad \text{Load}^{c,p,k} \in \bigcup (\text{Time Window}_c - T_c) \quad (30)$$

$$\forall c k \quad \text{size}(\text{Trip}^{c,p,k}) = 2 \times T_c + 2 \times \text{size}(\text{Load}^c) \quad (31)$$

$$\forall c p k \quad \text{alternative}(\text{Trip}^{c,p,k}, \{\text{Trip}_t^{c,p,k} \mid \forall t\}) \quad (32)$$

$$\forall t \quad \text{noOverlap}(\{\text{Trip}_t^{c,p,k} \mid \forall c p k\}) \quad (33)$$

$$\forall t \quad \text{Truck}_t = \text{span}(\{\text{Trip}_t^{c,p,k} \mid \forall c p k\}) \quad (34)$$

$$\forall t \quad \text{size}(\text{Truck}_t) \leq 12 \text{ hours} \quad (35)$$

On reconnaît dans les équations (23 - 25) un flexible job-shop, (27 - 29) définissent une distance minimale entre les débuts de livraison pour un client c tandis que (30) leur impose des fenêtres temporelles. Les équations (31 - 35) sont un bin-packing temporel. La contrainte (26) impose un minimum de livraisons.

4 Modélisation des sous-problèmes

Une fois les approximations fixées (section 3.4), nous considérons les différents modèles possibles pour les sous problèmes identifiés.

4.1 Flexible jobshop avec modes

Considérons à nouveau les équations du sous-problème de sélection des lignes et des produits

$$\forall c k \text{ alternative}(\text{Load}_l^{c,k}, \{\text{Load}_l^{c,k} \mid \forall l\}) \quad (1)$$

$$\forall c k \text{ alternative}(\text{Load}_p^{c,k}, \{\text{Load}_p^{c,k} \mid \forall p\}) \quad (2)$$

$$\forall c k \text{ presence}(\text{Load}_l^{c,k}) == \sum_{l,p \in LP} \text{presence}(\text{Load}_l^{c,k}) \times \text{presence}(\text{Load}_p^{c,k}) \quad (3)$$

$$\forall c p \sum_k \text{presence}(\text{Load}_p^{c,k}) \leq N_p^c \quad (4)$$

$$\forall l \text{ noOverlap}(\{\text{Load}_l^{c,k} \mid \forall c k\}) \quad (5)$$

Ce problème peut être vu comme un flexible-jobshop avec

- un job par client
- une ressource principale, la ligne sur laquelle est exécutée la tâche de remplissage (1) avec une contrainte d'absence de chevauchement (5)
- une ressource secondaire, le type de produit (2)
- une contrainte de compatibilité entre les ressources primaires et secondaires (3)
- une contrainte de nombre total de ressources secondaires utilisées (4)

Les contraintes secondaires de cette forme sont parfois appelées **modes** et sont très courantes dans les problèmes réels de manufacture (outils, travailleurs, etc).

Plusieurs modèles sont possibles pour représenter un flexible jobshop avec modes

1. double alternative
2. alternatives hiérarchiques - ligne en premier
3. alternatives hiérarchiques - produit en premier
4. séparation par mode
5. alternative simple bidimensionnelle

4.1.1 Double alternative

Ce sont les équations (1-5). L'équation de compatibilité (3) se simplifie en vertu de la spécialisation des lignes :

$$\forall c k p \text{ presence}(\text{Load}_p^{c,k}) == \sum_{l \in L_p} \text{presence}(\text{Load}_l^{c,k}) \quad (36)$$

indiquant que si le produit p est choisi pour la charge (c, k) , elle doit avoir lieu dans une ligne compatible avec p .

Si on remplace (36) dans le décompte des produits (4), on constate ne plus avoir besoin des variables choisissant les produits. On retrouve le modèle d'alternative hiérarchique avec ligne en premier.

4.1.2 Alternatives hiérarchiques - ligne en premier

La double alternative peut être transformée en une alternative hiérarchique dans laquelle on choisit la ligne et conditionnellement à cette ligne, le type de produit.

$$\forall c k \text{ alternative}(\text{Load}_l^{c,k}, \{\text{Load}_l^{c,k} \mid l \in \text{lines}\}) \quad (37)$$

$$\forall c k l \text{ alternative}(\text{Load}_l^{c,k}, \{\text{Load}_{l,p}^{c,k} \mid p \in P_l\}) \quad (38)$$

$$\forall l \text{ noOverlap}(\{\text{Load}_l^{c,k} \mid \forall c \forall k\}) \quad (39)$$

$$\forall c p \sum_k \sum_{l \in L_p} \text{presence}(\text{Load}_{l,p}^{c,k}) \leq N_p^c \quad (40)$$

La contrainte (2) devient conditionnelle à la ligne l , la contrainte de compatibilité disparaît dans le pré-calcul de P_l . Sous hypothèse supplémentaire de spécialisation des lignes on a $P_l = \{p\}$ donc la seconde alternative disparaît entièrement

$$\forall c k \text{ alternative}(\text{Load}_l^{c,k}, \{\text{Load}_l^{c,k} \mid l \in \text{lines}\}) \quad (41)$$

$$\forall l \text{ noOverlap}(\{\text{Load}_l^{c,k} \mid \forall c \forall k\}) \quad (42)$$

$$\forall c p \sum_k \sum_{l \in L_p} \text{presence}(\text{Load}_l^{c,k}) \leq N_p^c \quad (43)$$

4.1.3 Alternatives hiérarchiques - produit en premier

Si on choisit le produit en premier on obtient

$$\forall c k \text{ alternative}(\text{Load}_p^{c,k}, \{\text{Load}_p^{c,k} \mid \forall p\}) \quad (44)$$

$$\forall c k p \text{ alternative}(\text{Load}_p^{c,k}, \{\text{Load}_{l,p}^{c,k} \mid l \in L_p\}) \quad (45)$$

$$\forall l \text{ noOverlap}(\{\text{Load}_{l,p}^{c,k} \mid \forall c k p\}) \quad (46)$$

$$\forall c p \sum_k \text{presence}(\text{Load}_p^{c,k}) \leq N_p^c \quad (47)$$

Dans ce cas on a $k \in [1 \dots \sum_p N_p^c]$ et il y a $(\sum_p N_p^c) \times |P|$ variables de type $\text{Load}_p^{c,k}$. On peut réduire le nombre de variables en séparant les indexes $k \in [1 \dots \sum_p N_p^c]$ par type de produit, donnant lieu à la séparation par modes

4.1.4 Séparation par modes

Si on sépare l'indice $k \in [1 \dots \sum_p N_p^c]$ en $\forall p k_p \in [1 \dots N_p^c]$, et si on fixe le type de produit p pour chacun de ces nouveaux indices, on obtient le modèle

$$\forall c p k \text{ alternative}(\text{Load}_l^{c,p,k}, \{\text{Load}_l^{c,p,k} \mid l \in L_p\}) \quad (48)$$

$$\forall l \text{ noOverlap}(\{\text{Load}_l^{c,p,k} \mid \forall c k p\}) \quad (49)$$

4.1.5 Alternative bidimensionnelle

On peut enfin faire une alternative simple sur un indice bi-dimensionnel représentant les couples ligne $\times$ produit admissibles.

$$\forall c k \text{ alternative}(\text{Load}_l^{c,k}, \{\text{Load}_{l,p}^{c,k} \mid (l,p) \in LP\}) \quad (50)$$

$$\forall l \text{ noOverlap}(\{\text{Load}_{l,p}^{c,k} \mid \forall c k p \in P_l\}) \quad (51)$$

$$\forall c p \sum_k \sum_l \text{presence}(\text{Load}_{l,p}^{c,k}) \leq N_p^c \quad (52)$$

Si chaque ligne a un seul produit, le produit p devient une fonction de l et les équations deviennent identiques aux équations que l'alternative hiérarchique qui commence par choisir la ligne, après simplification en vertu de l'approximation de ligne spécialisée (41 - 43)

4.2 Rupture de symétrie et noOverlap

Les ruptures de symétrie sont des **dominations** : ce sont des approximations, donc elles éliminent des solutions mais gardent un représentant de chaque classe de solutions si bien qu'au moins une solution optimale est conservée.

Quand les intervalles impliqués dans une contrainte noOverlap sont identiques, on peut introduire un ordre entre eux et éliminer la contrainte de non chevauchement. Deux approches sont possibles

Précédences

$$\forall c \forall i < j \quad \text{end}(\text{Load}^{c,i}) \leq \text{start}(\text{Load}^{c,j})$$

chaînes

$$\begin{aligned} \forall c \ k \quad \text{end}(\text{Load}^{c,k}) &\leq \text{start}(\text{Load}^{c,k+1}) \\ \forall c \ k \quad \text{presence}(\text{Load}^{c,k+1}) &\Rightarrow \text{presence}(\text{Load}^{c,k}) \end{aligned}$$

4.2.1 Impact de la rupture de symétrie sur les équations de flexible jobshop

Le modèle de séparation par modes (section 4.1.4) utilise les variables $\text{Load}^{c,p,k}$. On peut poser une rupture de symétrie par couple (c, p) mais il restera une contrainte noOverlap pour l'ensemble des produits

En revanche, en absence de rupture de symétrie, le modèle *alternative hiérarchique - ligne en premier* (section 4.1.2) et le modèle *séparation de modes* (section 4.1.4) se confondent car on peut artificiellement ordonner les variables $\text{Load}^{c,k}$ en décidant que les N_p^c premières seront attribuées le produit p , etc.

Ces modèles ne sont différents que si on pose des précédences sur $\text{Load}^{c,k}$ de la forme $\text{end}(\text{Load}^{c,k}) \leq \text{start}(\text{Load}^{c,k+1})$. De manière plus générale une séquence $\text{Load}^{c,0} \text{ Load}^{c,1} \dots \text{Load}^{c,k}$ n'implique pas nécessairement des précédences entre ces tâches. Et on peut exploiter cette liberté ou bien pour imposer une chaîne de précédences, ou bien pour fixer les produits de certaines tâches.

4.3 Fenêtres temporelles

4.3.1 Fenêtres et chaînes

Quand les décharges forment une chaîne et doivent avoir lieu à l'intérieur de fenêtres temporelles, on peut raffiner les temps de début de certaines décharges dans la chaîne. Par exemple dans la figure 8 la décharge 5 ne peut avoir lieu dans la première fenêtre temporelle.



FIGURE 8 – Deux fenêtres temporelles et 6 décharges

4.3.2 Disjonction sur les fenêtres

Considérer une disjonction sur les fenêtres de livraison, c'est à dire un intervalle par livraison par fenêtre semble raisonnable dans la mesure où (i) seule la moitié des clients a des fenêtres de livraison et (ii) dans le jeu de données existant il n'y a que deux fenêtres (9h-12h et 14h-16h).

5 Extension à plusieurs journées

Nous étendons maintenant le modèle journalier à 7 jours, en séparant conducteurs et camions. Les contraintes portant sur les conducteurs sont

- nb_conducteurs conducteurs
- nb_camions camions
- 5 journées de travail (shifts) de 8h par conducteur
- 2 jours de repos consécutifs par conducteur
- 12 heures entre shifts
- au plus un shift par jour
- les shifts commencent et terminent à l'heure ronde

5.1 Représentation des shifts

Étendre le modèle journalier à plusieurs jours exige de revisiter le sous problème de transport (14 - 17) qui garantit que les voyages ont lieu à l'intérieur des shifts des camion/conducteur choisis. En particulier la contrainte span qui garantissait que les voyages aient lieu à l'intérieur du shift journalier du camion doit être étendue à plusieurs shifts.

5.1.1 Alternative sur les conducteurs $\times$ shifts

Considérons les variables

- Shift_d^s représentant le s -ième shift $s \in [1 \dots 5]$ du conducteur d
- $\text{Trip}_{d,s}^{c,k}$ représentant le fait que le voyage (c, k) est effectué par le conducteur d dans son shift s

Les équations suivantes remplacent les équations (14 - 17)

$$\forall c \forall k \quad \text{alternative}(\text{Trip}^{c,k}, \{\text{Trip}_{d,s}^{c,k} \mid \forall d \forall s\}) \quad (53)$$

$$\forall d \forall s \quad \text{Shift}_s^d = \text{span}(\{\text{Trip}_{d,s}^{c,k} \mid \forall c \forall k\}) \quad (54)$$

$$\forall d \forall s \quad \text{noOverlap}(\{\text{Trip}_{d,s}^{c,k} \mid \forall c \forall k\}) \quad (55)$$

Cette représentation crée nb_voyages $\times$ nb_shifts $\times$ nb_conducteurs intervalles pour représenter les voyages

5.1.2 noOverlap avec anti-shifts

Une seconde approche évite l'alternative en introduisant une contrainte noOverlap entre les tâches représentant le complémentaire des shifts (les anti-shift) et les voyages (figure 9)

$$\forall c \forall k \quad \text{alternative}(\text{Trip}^{c,k}, \{\text{Trip}_d^{c,k} \mid \forall d\}) \quad (56)$$

$$\forall d \forall s \quad \text{end}(\text{Anti_shift}_s^d) = \text{start}(\text{Shift}_s^d) \quad (57)$$

$$\forall d \forall s \quad \text{end}(\text{Shift}_s^d) = \text{start}(\text{Anti_shift}_{s+1}^d) \quad (58)$$

$$\forall d \quad \text{noOverlap}(\{\text{Anti_shift}_s^d \mid \forall s\} \cup \{\text{Trip}_d^{c,k} \mid \forall c \forall k\}) \quad (59)$$

Cette représentation crée nb_voyages $\times$ nb_conducteurs intervalles pour représenter les voyages



FIGURE 9 – tâches entre les anti-shifts en noir.

5.2 Modèle multi-jours

Nous pouvons à présent écrire un modèle pour une semaine

Variables

- $\text{Load}^{c,p,k}$ la charge optionnelle de la k -ième livraison du produit p pour le client c
- $\text{Load}_l^{c,p,k}$ la charge optionnelle sur la ligne l de la k -ième livraison du produit p pour le client c
- $\text{Trip}^{c,p,k}$ la k -ième livraison optionnelle du produit p pour le client c
- $\text{Trip}_t^{c,p,k}$ la k -ième livraison optionnelle du produit p pour le client c opéré sur le camion t
- $\text{Shift}^{d,s}$ le s -ième shift du conducteur d
- $\text{Anti_shift}^{d,s}$ l'espace entre les shifts $s - 1$ et s pour le conducteur d
- $\text{Shift_day}^{d,s}$ intervalle indiquant la journée dans laquelle se trouve le shift s du conducteur d
- Off^d sequence de 2 jours de repos pour le conducteur d

Équations

$$\max \sum_{c,p,k} \text{presence}(\text{Trip}^{c,p,k})$$

$$\forall c \forall p \forall k \quad \text{alternative}(\text{Load}^{c,p,k}, \{\text{Load}_l^{c,p,k} \mid \forall l \in L_p\}) \quad (60)$$

$$\forall l \quad \text{noOverlap}(\{\text{Load}_l^{c,p,k} \mid \forall c \forall p \in P_l \forall k\}) \quad (61)$$

$$\forall c \forall p \forall k \quad \text{Load}_l^{c,p,k} \in \bigcup \text{Operating hours}_l \quad (62)$$

$$\forall c \forall k \quad \text{size}(\text{Trip}^{c,k}) = 2 \times (T_c + \text{size}(\text{Load}^c)) \quad (63)$$

$$\forall c \forall p \forall k \quad \text{start}(\text{Trip}^{c,p,k}) = \text{start}(\text{Load}^{c,p,k}) \quad (64)$$

$$\forall c \quad \text{noOverlap}(\{\text{Load}^{c,p,k} \mid \forall p \forall k\}) \quad (65)$$

$$\forall c \forall p \forall k \quad \text{Load}^{c,p,k} \in \bigcup \text{Time Window}_c - T_c \quad (66)$$

$$\forall c \forall k \quad \text{alternative}(\text{Trip}^{c,k}, \{\text{Trip}_d^{c,k} \mid \forall d\}) \quad (67)$$

$$\forall d \forall s \quad \text{end}(\text{Anti_shift}_s^d) = \text{start}(\text{Shift}_s^d) \quad (68)$$

$$\forall d \forall s \quad \text{end}(\text{Shift}_s^d) = \text{start}(\text{Anti_shift}_{s+1}^d) \quad (69)$$

$$\forall d \forall s \quad \text{size}(\text{Anti_shift}_s^d) \geq 12 \text{ hours} \quad (70)$$

$$\forall d \forall s \quad \text{size}(\text{Shift}_s^d) \leq 7 \text{ hours} \quad (71)$$

$$\forall d \quad \text{noOverlap}(\{\text{Anti_shift}_s^d \mid \forall s\} \cup \{\text{Trip}_d^{c,k} \mid \forall c \forall k\}) \quad (72)$$

$$\forall d \forall s \quad \text{start}(\text{Shift_day}_s^d) = \text{start}(\text{Shift}_s^d) \text{ div } 24 \text{ hours} \quad (73)$$

$$\forall d \quad \text{noOverlap}(\{\text{Shift_day}_s^d \mid \forall s\} \cup \{\text{Off}^d\}) \quad (74)$$

$$\forall d \forall s \quad \text{size}(\text{Shift}_s^d) = 1 \quad (75)$$

$$\forall d \quad \text{size}(\text{Off}^d) = 2 \quad (76)$$

$$\sum_d \sum_s \text{pulse}(\text{Shift}_s^d) \leq \text{nb_trucks} \quad (77)$$

Les équations (60 - 66) définissent le flexible jobshop comme auparavant. L'équation (67) impose que pour tout voyage soit choisi un conducteur - et non plus un camion comme auparavant. Les équations (68 - 69) construisent la séquence de shifts et anti-shifts, tandis que les équations (70) et (71) imposent des contraintes sur la durée des shifts

et la durée des espaces entre les shifts (les anti-shifts). La contrainte (72) impose que les voyages et les anti-shifts ne se chevauchent pas, autrement dit que les voyages se placent quelque part dans les shifts / entre les anti-shifts. L'équation (73) synchronise les shifts et leur représentation modulo 24h tandis que l'équation (74) impose un non chevauchement entre les journées travaillées et les deux jours de repos consécutifs. Les contraintes (75) et (76) fixent la durée en jours des journées de travail et de repos. Enfin la contrainte (77) impose qu'à tout instant il n'y ait pas plus de shifts de conducteurs que de camions disponibles.

5.3 Taille du modèle multi-jours

En théorie c'est au modèle de déterminer combien de conducteurs sont nécessaires pour les opérations mais un nombre variable de conducteurs s'accommode mal avec la représentation en shifts et anti-shifts. A la place nous faisons un calcul de dimensionnement de la forme suivante

- dans le modèle journalier il y a $H = \text{nb_camions} \times 12h$ heures de travail
- dans le modèle multi-jour il y a $M = \text{nb_conducteurs} \times \text{nb_jours_de_travail} \times 7h$ heures de travail
- on pose $7 \times H = M$ pour un modèle sur 7 jours

Numériquement on obtient $c = \frac{7 \times H}{M/c} = \frac{7 \times 61 \times 12}{5 \times 7} = 147$ conducteurs pour 2000 livraisons. L'alternative qui permet d'allouer une livraison à un conducteur (67) crée donc 300 000 intervalles, ce qui est à la limite de ce que peuvent supporter les moteurs d'ordonnancement comme OptalCP ou CPO. Donc le problème multi-jour est encore trop grand.

6 Expérimentations

6.1 Données de test

Les données contiennent les ordres de livraison pour 13 journées indépendantes (entre 43 et 373 livraisons par jour) tout comme les clients qui chargent leurs hydrocarbures sur leurs propre flotte. La journée 10 ne compte que 43 livraisons car c'est un jour férié où seul 1 shift de 8h de travail est possible mais nous la résolvons avec les mêmes contraintes que les autres pour l'instant.

La séparation en 13 journées est due à l'heuristique de fenêtres glissantes de la solution en place. Les données brutes pour un horizon de 14 jours sont les ordres de livraison des 13 journées concaténés, et tout autre partition de ces données est un jeu de test valide.

Ces données et les modèles testés sont disponibles en ligne à l'adresse <https://optalcp.com/docs/benchmarks/> sous le nom tank-truck

6.2 Modèles testés

Nous avons considéré

1. **Approximations**
 - (a) réduction de 2 cuves à une
 - (b) élimination du temps d'attente
 - (c) spécialisation des lignes
2. **Ruptures de symétrie pour les chargements**

- (a) noOverlap (pas de rupture de symétrie)
- (b) précédences
- (c) chaînes (traitement spécial par OptalCP)
- (d) chaînes suivant la définition

3. Représentations du flexible jobshop

- (a) double alternative
- (b) alternatives hiérarchiques - ligne en premier
- (c) alternatives hiérarchiques - produit en premier
- (d) alternative bidimensionnelle
- (e) séparation par modes

4. Représentations des fenêtres temporelles

- (a) calendriers fournis par le moteur (forbiddenExtent)
- (b) bornes en présence de chaînes
- (c) disjonction sur la fenêtre

5. Représentations du sous problème de bin-packing

- (a) alternatives
- (b) alternatives + contrainte pack redondante

Par ailleurs, en fonction des approximations utilisées certaines de ces représentations fusionnent (notamment les modèles pour le flexible jobshop avec la spécialisation des lignes ou l'absence de rupture de symétrie).

Nous ne présenterons ici que les résultats pour deux modèles (1abc2a3b4a5a) et (1abc2c3e4a5a) autrement dit toutes les approximations et d'un côté l'alternative hiérarchique ligne en premier avec rupture de symétrie par chaîne, de l'autre le modèle par séparation de modes sans rupture de symétrie. Nous les nommons **3b** et **3e** respectivement.

6.3 Moteurs testés

Nous n'avons pas l'autorisation de présenter les résultats obtenus par la solution en place chez le client. A défaut nous comparons **OptalCP** 2024.11.1.20 contre **ILOG CP Optimizer** 22.1.1

Les test on eu lieu en 300 secondes sur un laptop 4 cores i7-1185G7@3.00GHz avec 16GB de RAM.

6.4 Résultats

Quelques solutions des instances journalières pour OptalCP et CPO en 5 minutes.

Les bornes inférieures (autrement dit d'autres solutions au problème de maximisation) de la colonne LB ont été trouvées en laissant OptalCP tourner pendant 10-30 minutes pour les instances 5,6,15,13,11 et une semaine pour les instances 4 et 12. Les bornes supérieures sont égales au nombre d'ordres de livraison, aucun moteur n'ayant pu démontrer qu'il n'est pas possible de livrer tous les ordres.

Le modèle sans rupture de symétrie obtient de meilleurs résultats.¹ Tous les autres variantes de modèles ont des résultats significativement inférieurs, et les 3 approximations réduisent grandement le problème par rapport au modèle de référence (a 2 cuves).

1. Une version antérieure de cette étude montrait des résultats plus proches mais les améliorations dans OptalCP ont bénéficié davantage **3e**.

jour	livraisons	LB	UB	OptalCP		CPO
				3b	3e	3e
10	43	43	43	43	43	43
9	294	294	294	294	294	294
16	319	319	319	317	319	317
8	322	322	322	322	322	320
14	331	331	331	331	331	326
7	341	341	341	341	340	336
5	344	341	341	337	339	329
6	344	344	344	338	344	305
15	349	349	349	339	348	334
13	358	358	358	346	353	339
11	366	366	366	349	352	317
4	371	369	371	339	353	344
12	373	367	373	342	357	339

TABLE 1 – Résultats de OptalCP et CPO sur les 13 journées considérées indépendamment. En gras le modèle+moteur ayant obtenu la meilleure solution dans ce test pour une instance donnée.

7 Conclusion

Nous avons résolu un problème de transport d'hydrocarbures par camion citerne avec le moteur d'ordonnancement OptalCP. Le problème est un exemple de problème réel complexe où sont mélangés des aspects de flexible-jobshop, de tournée de véhicules avec fenêtres, d'emplois du temps de personnel et de gestion d'inventaire. Après analyse de chaque sous-problème nous avons proposé diverses approximations et modélisations que nous avons ensuite testées. Un jeu d'instances public a été créé afin que d'autres équipes puissent également tester leurs approximations et modèles. Il nous semble important de souligner une fois de plus qu'il s'agit d'un problème dont les contours ne sont pas fixés et que toutes les approximations et modèles qui en pratique aboutissent à une amélioration sont acceptables.

Références

- [1] Diego Olivier Fernandez Pons and Petr Vilím. OptalCP un nouveau moteur d'ordonnancement. *Journées Francophones de Programmation par Contraintes 2024 (JFPC 2024)*.
- [2] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling : 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, 23 :210–250, 2018.
- [3] Pierre Schaus, Yves Deville, et al. A global constraint for bin-packing with precedences : Application to the assembly line balancing problem. In *AAAI*, pages 369–374, 2008.
- [4] Paul Shaw. A constraint for bin packing. In *International conference on principles and practice of constraint programming*, pages 648–662. Springer, 2004.

